TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: M 2612 – Elektrotechnika a informatika

Studijní obor: 3906T001 – Mechatronika

Programování Direct3D v programovacím prostředí Borland Delphi

Programming Direct3D in programming environment Borland Delphi

Diplomová práce

Autor: **Jaroslav Hlavatý**

Vedoucí diplomové práce: Ing. Tomáš Pluhař

Konzultant: Ing. Tomáš Martinec

Liberec 18.5.2007

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé diplomové práce panu Ing. Tomáši Pluhařovi, který byl vždy ochoten konzultovat problémy týkající se mé diplomové práce. Také bych chtěl poděkovat svým rodičům za podporu, které se mi od nich dostávalo po celou dobu studia.

Abstrakt

Diplomová práce se zabývá možnostmi knihoven DirectX, speciálně Direct3D. Cílem je nastudování knihovny Direct3D a vytvoření několika programů demonstrujících jejich použití. Důraz je kladen na přehlednost a jednoduchost vyvinutých aplikací. Diplomová práce je psána jako příručka programátora. Závěrečná část je věnována porovnání knihoven Direct3D a OpenGL.

Klíčová slova: DirectX, Direct3D, Delphi, SDK, grafika

Abstract

The diploma thesis deals with opportunities of DirectX libraries, especially Direct3D. The Aim is to study the Direct3D libraries and to create several programmes demonstrating their usage. Lucidity and simplicity of developed applications are emphasized. The thesis is written like a guidebook for programmers. The final part is devoted to the comparison of Direct3D and OpenGL libraries.

Keywords: DirectX, Direct3D, Delphi, SDK, graphics

Obsah

Seznam ilustrací

Seznam použitých termínů a zkratek

API	*Application Programing Interface* – Rozhraní pro programování aplikací.
ARGB	Makro, pracující s hodnotou alfa a intenzitou jednotlivých složek RGB.
CAD	*Computer Aided Design* – Počítačem podporované navrhování – zkratka označující software pro projektování či konstruování na počítači.
COM	*Component Object Model* – Hlavní objektová technologie operačních systémů Windows. V současné době nejrozšířenější objektový model.
Direct3D	Komponenta knihovny DirectX pro grafiku a 3D vykreslování.
DirectX	Knihovna obsahující nástroje pro tvorbu počítačových her a dalších multimediálních aplikací vytvořená firmou Microsoft.
EAX	*Enviromental Audio Extension* – Technologie, která umožňuje umístit zvukový zdroj v prostoru prostřednictvím parametrů vystihující základní zvukové vlastnosti jako rezonance nebo míra pohlcení zvuku.
FPS	*Frame Per Second* – Počet zobrazených snímků za sekundu.
GDI	*Graphics Device Interface* – Sada standardních procedur systému Windows pro práci s grafikou.
OpenGL	*Open Graphics Library* - Průmyslový standard specifikující rozhraní pro tvorbu aplikací počítačové grafiky.
RGB	Barevný model značící červenou, zelenou a modrou barvu.
RHW	Reciprocita homogenního W.
SDK	*System Development Kit* – Soubor základních nástrojů pro vývoj aplikací v DirectX, nabízený firmou Microsoft.
Vertex	Anglický výraz pro vrchol.
XRGB	Makro, pracující s intenzitou jednotlivých složek RGB.

Úvod

Psaní programů imitující skutečný svět není vůbec jednoduché. Je k tomu třeba znát spoustu znalostí hned z několika vědních disciplín současně. Avšak i přes tento nelehký úkol se začali tisíce programátorů věnovat modelování skutečného světa pomocí několika vzorců a algoritmů a vytvořili tak ucelenou 3D-knihovnu nazvanou Microsoft DirectX. Díky tomuto balíku lze vytvářet virtuální svět bez hlubších znalostí matematiky či fyziky. Hlavní částí balíku DirectX je knihovna Direct3D, která obsahuje nástroj pro tvorbu trojrozměrných objektů a scén a nabízí ho všem, kteří mají zájem tento virtuální svět vytvářet.

Diplomová práce je určena jak studentům informačních technologií, tak i každému, kdo má zájem o pochopení principů pro modelování trojrozměrných objektů a scén ve virtuální realitě. Dále by měla sloužit všem, kteří se chtějí snadno a rychle naučit používat grafické techniky knihovny Direc3D programového balíku DiectX v programovacím prostředí Borland Delphi. Důležitou věcí při vytváření všech aplikací je, aby byly jednoduché a srozumitelné. To umožní následné snadné použití při programování kompletního programu imitujícího skutečný svět. Na těchto aplikacích jsou demonstrovány základní metody a techniky programování virtuálních scén v Direct3D. Uživateli by diplomová práce měla sloužit jako příručka programátora.

Po formální stránce je diplomová práce členěna do dvanácti kapitol. Úvodní část je věnována teoretickým poznatkům, vycházejícím ze studia dostupné literatury. Praktická část popisuje tvorbu jednoduchých virtuálních objektů a scén. Tato část začíná prvotním nastavením a inicializací Direct3D a postupně se věnuje vykreslování základních primitiv, transformacím, používání nasvícení, mapování textur, použití průhlednosti a mlhy ve scéně nebo vykreslení složitějších tzv. drátěných objektů exportovaných ze speciálního softwaru pro tvorbu těchto objektů, jako je například 3D Studio Max. Závěrečná část práce se věnuje porovnání grafické knihovny Direct3D s knihovnou OpenGL.

1 DirectX

DirectX je pokročilá sada multimediálních rozhraní, vyvinutá firmou Microsoft, pro programování grafických aplikací (API) pod operačním systémem Windows. Jinými slovy je to programátorská knihovna obsahující nástroje pro tvorbu počítačových her a dalších multimediálních aplikací. Poskytuje standardní vývojovou platformu pro Windows a dává možnost softwarovým vývojářům k tomu, aby zpřístupnili speciální vlastnosti hardwaru bez toho, aby museli psát strojový kód. Tato technologie byla prvně představena v roce 1995 a je standardizována pro vývoj aplikací založených na platformě Windows.

DirectX je technologie, která umožňuje vyšší výkon v grafice a zvuku pro hraní her či sledování videa na PC.

V jádru jsou to jeho rozhraní pro programování aplikací neboli API. API působí jako most mezi hardware a software pro vzájemnou komunikaci. API umožňuje multimediálním aplikacím přístup k pokročilým vlastnostem výkonného hardwaru jako jsou 3D grafické akcelerátory a zvukové karty. Ovládají funkce nižší úrovně, včetně 2D grafických akcelerátorů. Zajišťují podporu vstupních zařízení jako jsou joysticky, klávesnice a myši, a kontrolu nad mícháním a výstupem zvuku. Díky DiretX je docíleno lepší grafiky a zvukových efektů na počítačích.

1.1 Komponenty DirectX

DirectX se skládá z několika částí. Nejdůležitější částí a částí, kterou se diplomová práce zabývá je knihovna Direct3D. Zde popsány jednotlivé části DirectX.

- *DirectX Graphics* (a jeho části *DirectDraw* a *Direct3D*) – Soubor tříd a funkcí schopných rychlého vykreslování do grafické paměti. Umožňuje modelování 3D objektů v prostoru.
- *DirectInput* – Součást, která podporuje v aplikaci použít vstupní zařízení jako např. myši, klávesnice, joysticky, gamepady apod. včetně podpory technologie force feedback.
- *DirectPlay* – Tato část slouží ke spojení naší aplikace s jiným počítačem nebo s Internetem (pomocí protokolu TCP/IP). Podporuje hry více hráčů po síti.

- *DirectSound* – Knihovna zabývající se oblastí zvuku, neboli jak relativně jednoduše implementovat do programu zvukovou kulisu. Podporuje přehrávání a záznam zvuků.
- *DirectMusic* – S touto komponentou lze přehrávat a zpracovávat hudbu i zvuky. Podporuje EAX (Enviromental Audio Extension) a 3D zvuk.
- *DirectShow* – Komponenta pro podporu multimediálních aplikací, přehrávání a zpracování videa a zvuku.
- *DirectSetup* – Jednoduchý nástroj umožňující instalaci knihovny DirectX na PC.
- *DirectX Media Objects* – Nástroj pro tvorbu multimediálních efektů a kodeků.

Nelze přehlédnout, že všechny komponenty obsahují slůvko *Direct* neboli *Přímo*, což je přesně to, co DirectX dělá. DirectX totiž zapisuje přímo do grafické paměti a přímo do zvukové paměti. DirectX pracuje na nejnižší úrovni tzn. na úrovni hardwaru, a právě proto je o tolik rychlejší než GDI.

1.2 Úvod do knihovny Direct3D

Direct3D je programovací rozhraní pro vytváření trojrozměrných virtuálních aplikací. Díky Direct3D lze provádět cokoliv od zobrazování jednoduchých tvarů přes vytváření trojrozměrných scén až po skládání animovaných scén s použitím mapování textur, vyhlazováním či nasvícením. Většina související matematiky je ukryta do knihoven, a proto se programátor nemusí zabývat složitými výpočty popisující chování objektů ve scéně nebo různým efektům, jako jsou například posun a rotace objektů nebo definování vlastností světel.

1.3 Knihovna Direct3D

Již bylo zmíněno, že Direct3D je jednou ze součástí programovacího balíku DirectX, který kromě programování grafiky nabízí metody pro práci se zvukem, vstupními zařízeními, sítí apod. Předchozí verze DiectX obsahovaly kromě knihovny Direct3D pro vývoj trojrozměrných aplikací i knihovnu DirectDraw pro vývoj aplikací dvourozměrných. Od verze Microsoft DirectX 8 jsou tyto dvě knihovny sloučeny v jednu pojmenovanou DirectX Graphics.

Direct3D nám nabízí velké množství rozhraní COM, z nichž každé reprezentuje objekt Direct3D. Hlavní část Direct3D zahrnuje přibližně 20 různých rozhraní COM. Příklad jednoho takového rozhraní je *IDirect3D9*, které můžeme použít k vytváření

objektů zařízení, dotazování na zařízení, řízení režimů grafických adaptérů, dotazování na vlastnosti zařízení a další. Objekt *IDirect3D9* lze vytvořit zavoláním metody *Direct3DCreate()*. Vytvářet můžeme i jiné typy objektů z rozhraní nižší hierarchie. Nejčastěji používaná rozhraní Direct3D podle pořadí, ve kterém jsou objekty vytvářeny jsou uvedena zde:

- *IDirect3D9* – Představuje hlavní objekt Direct3D, který je počátečním bodem pro všechny aplikace Direct3D.
- *IDirect3DDevice* – Představuje zařízení Direkt 3D, které je druhým nejvýše postaveným objektem v hierarchii Direct3D. Velká část programování v Direct3D je prováděna právě prostřednictvím tohoto rozhraní.
- *IDirect3DSurface* – Představuje povrch Direct3D, tedy obecně řečeno oblast paměti, ve které jsou grafická data uložena.
- *IDirect3DTexture9* – Představuje texturu Direct3D, tedy obrázek, který je možné nanést na trojrozměrný objekt za účelem zvýšení detailnosti povrchu.
- *IDirect3DVertexBuffer9* – Představuje Direct3D vertex buffer, tedy oblast paměti, která uchovává souřadnice bodů, jenž přestavují trojrozměrné tvary.
- *IDirect3DIndexBuffer9* – Představuje Direct3D index buffer, který uchovává odsazení dat ve vetex bufferu.

1.4 Nejčastěji používané funkce Direct3D

Pomocí Direct3D lze tvořit mnoho typů trojrozměrných grafických objektů a efektů. Vykreslováním jednoduchých polygonů počínaje a vytváření trojrozměrných animovaných scén konče. V následující části jsou shrnuty nejčastěji používané funkce grafické knihovny Direct3D.

- *Kreslení objektů* – Direct3D dokáže vykreslit body, čáry a trojúhelníky, pomocí kterých lze vytvořit jakýkoliv trojrozměrný objekt. Polygony (trojúhelníky) jsou tvořeny množinou vrcholů (VERTEXů).
- *Prohlížení objektů* – Po složení objektů ve scéně, je třeba Direct3D říct, jak mají tyto objekty být zobrazeny. Toho dosáhneme zavoláním metod Direct3D, které provádí několik typů transformací (transformace objektu, pohledu, projekce).
- *Používání nasvícení* – Na obrazovce se vykreslené tvary nezobrazí, pokud do scény nebude přidáno světlo. Kromě toho, že Direct3D dokáže pracovat

s mnoha typy světelných zdrojů (okolní, rozptýlené, přímé a odražené světlo), umožňuje nám také určit vlastnosti materiálu, od kterých se světlo odráží.

- *Vylepšení obrazu* – Direct3D také nabízí několik metod pro přidání reality do trojrozměrných scén. Tyto metody používají k vylepšení obrazu techniky, jako je vyhlazování hran, míchání nebo mlha. Vyhlazování hran upravuje přechody hran, míchání umožňuje vytvářet průsvitné objekty a mlha způsobuje, že se objekty dále od kamery ztrácí v oparu.
- *Manipulace s bitmapami* – Direct3D nabízí několik způsobů manipulace s bitmapami. Například je možné obrázky přenášet mezi obrazovkou a pamětí.
- *Mapování textur* – Objekty vytvořené z polygonů postrádají detaily skutečných objektů. Z tohoto důvodu nabízí Direct3D programátorovi mapování textur.
- *Animace* – Direkt3D implementuje *multibuffering*, pomocí kterého je v paměti vytvořeno několik základních scén. Poté zobrazíme jednu scénu na obrazovku, zatímco s ostatními pracujeme v paměti. Je-li připravena další scéna, odešle se na obrazovku. Tím se uvolní místo v paměti a lze tak pracovat s dalším scénou.

1.5 Nastavení programovacího prostředí

Součástí programovacího prostředí Borland Delphi nejsou žádné unity, které by poskytovaly rozhraní do grafické knihovny Direct3D. Microsoft je nepíše v Delphi, nýbrž v programovacím jazyku C++. V současné době jsou ovšem přeprogramované *unity* pro programování grafiky v Direct3D v programovacím prostředí Delphi k dispozici na internetu. Tyto unity si lze stáhnou na internetových stránkách [3] nebo [11]. Po přidání těchto *unit* do programovacího prostředí Delphi je dále třeba nakopírovat potřebné dynamické knihovny do systému Windows. Těmito knihovnami jsou *d3dx9_31.dll* a *DXErr9ab.dll*, které lze stáhnou na internetových stránkách [3]. Všechny potřebné *unity* a dynamické knihovny, k programování v Direct3D v programovacím prostředí Borland Delphi, jsou také k dispozici na přiloženém CD.

Direct3D je určeno pro programování v programovacím prostředí C++. Veškerá dokumentace pro psaní grafických aplikací je optimalizována právě pro toto prostředí.

Zvolené programovací prostředí pro vývoj aplikací je Borland Delphi 7.0. Verze rozhraní DirecX je DirectX 9.0c. Rozhraní je součástí balíku SDK DirectX, který mimo jiné obsahuje kompletní dokumentaci knihovny DirectX. Tento vývojářský balík lze stáhnout na internetových stránkách [7].

2 Inicializace Direct3D

Všechny programy v Direct3D musí před vykreslováním na obrazovku inicializovat Direct3D. S tímto jevem se jako programátoři setkáme při psaní všech aplikací v Direct3D. Tato část je tedy jednou z nejdůležitějších pro psaní aplikací v Direct3D. Jednotlivě se věnuje následujícímu:

- Jak vytvořit objekt Direkt3D.
- Jak vytvořit objekt zařízení Direct3D.
- Režimům zobrazení a pixelovým formátům.
- Jak vykreslit zobrazení.
- Jak vyprázdnit obrazovku Direct3D.

2.1 Vytvoření objektu Direct3D

Prvním krokem při psaní programu v Direct3D je vytvoření objektu Ditect3D. Objekt poskytuje přístup k rozhraní IDirect3D. Toto rozhraní obsahuje metody, které jsou potřeba ke spuštění aplikací v Direct3D, ale také zahrnuje další metody, například pro kontrolu režimů zobrazení, vytváření zařízení Direct3D a další. Dále je ukázáno, jak některé metody volat. Nejdříve je však uvedeno, jak se objekt Direct3D vytváří:

```
g_pDirect3D         : IDIRECT3D9;
. . .
g_pDirect3D := Direct3DCreate9(D3D_SDK_VERSION);
```

V této části kódu je nejdříve deklarovaná globální proměnná *g_pDirect3D* typu *IDIRECT3D*. Poté je v implementační části zavolána metoda *Direct3DCreate()*, která je v API Direct3D deklarována následovně:

```
IDirect3D9* Direct3DCreate9(UNIT SDKVersion);
```

Funkce *Direct3DCreate()* vyžaduje jediný argument, pro který existuje jediná možnost: D3D_SDK_VERSION.

2.2 Vytvoření zařízení Direct3D

Druhým krokem při programování v Direct3D je vytvoření zařízení Direct3D, které je jedním z objektů rozhraní *IDirect3DDevice9*. Program může pomocí tohoto objektu přistupovat ke spoustě metod pro práci s grafickými zdroji, kreslení tvarů, manipulaci s obrázky a texturami a podobně. Téměř všechny grafické operace

v aplikacích Direct3D jsou prováděny prostřednictvím objektu *IDirect3DDevice9*. Vytvoření zařízení Direct3D je ukázáno zde:

Výpis 2.1 Vytvoření zařízení Direct3D

```
g_pZarizeniDirect3D  : IDirect3DDevice9;
. . .
g_pZarizeniDirect3D :=nil;
hr:=g_pDirect3D.CreateDevice(
  D3DADAPTER_DEFAULT,
  D3DDEVTYPE_HAL,
  Handle,
  D3DCREATE_SOFTWARE_VERTEXPROCESSING,
  @D3DParametry,
  g_pZarizeniDirect3D);
if FAILED(hr) then Exit;
```

Nejdříve je potřeba deklarovat proměnnou *g_pZařízeníDirect3D*, která je typu *IDirect3DDevice9*. Tuto proměnnou musíme inicializovat stejně jako v předchozím případě inicializační hodnotou NIL. Pro vytvoření objektu zařízení Direct3D se zavolá metoda *CreateDevice()* objektu Direct3D. V API je tato metoda deklarována následovně:

```
HRESULT CreateDevice(
  UINT Adapter,
  D3DDEVTYPE DeviceType,
  HWND hFocusWindow,
  DWORD BehaviorFlags,
  D3DPRESENT_PARAMETERS * pPresentationParameters,
  IDirect3DDevice9 ** ppReturnedDeviceInterface
);
```

Tato metoda vyžaduje šest parametrů, které jsou použity takto:

- *Adapter* – Číslo určující adaptér, pro který má Direct3D vytvořit zařízení.
- *DeviceType* – Definuje typ objektu, který se má použít.
- *hFocusWindow* – Manipulátor okna, se kterým bude zařízení asociováno.
- *BehaviorFlags* – Sada příznaků určujících požadované vlastnosti zařízení.
- *pPresentationParameters* – Ukazatel na instanci struktury D3DPRESENT_PARAMETERS.
- *ppReturnedDeviceInterface* – Adresa ukazatele na rozhraní *IDirect3DDevice9*.

Prvním argumentem metody *CreateDevice()* je adapter, který představuje grafické zařízení, které má uživatel v počítači nainstalované. Zpravidla tímto zařízením je grafická karta počítače.

Druhý argument určuje typ objektu zařízení, které má být vytvořeno. Knihovna Direct3D definuje 4 typy zařízení, z nichž poslední dvě nejsou v současné době využívána či podporována. Význam prvních dvou argumentů je následující:

- D3DDEVTYPE_HAL – Zařízení, které využívá 3D-hardware nainstalovaný v počítači.
- D3DDEVTYPE_REF – Zařízení, které vykonává funkce Direct3D softwarově.

Tyto typy zařízení mají vlastní konstantu definovanou ve struktuře D3DDEVTYPE (D3DDEVTYPE_HAL = 1, D3DDEVTYPE_REF = 2).

Dalším argumentem je manipulátor okna aplikace.

Čtvrtým argumentem je hodnota s příznaky, které určují chování zařízení. Celkem existuje sedm různých příznaků. Zmíněny jsou pouze dva: D3DCREATE_ HARDWARE_VERTEXPROCESING a D3DCREATE_SOFTWARE_VERTEX-PROCESING. První se používá k hardwarovému a druhý softwarovému zpracování vrcholů (VERTEXů). V této diplomové práci je použito příznaku D3DCREATE_ SOFTWARE_VERTEXPROCESING, který zajišťuje kompatibilitu se všemi systémy, za cenu mírně sníženého výkonu a efektivity.

Pátý argumentu je ukazatel na instanci struktury D3DPRESENT_ PARAMETRS. SDK DirectX definuje tuto strukturu následovně:

```
typedef struct D3DPRESENT_PARAMETERS {
  UINT BackBufferWidth, BackBufferHeight;
  D3DFORMAT BackBufferFormat;
  UINT BackBufferCount;
  D3DMULTISAMPLE_TYPE MultiSampleType;
  DWORD MultiSampleQuality;
  D3DSWAPEFFECT SwapEffect;
  HWND hDeviceWindow;
  BOOL Windowed;
  BOOL EnableAutoDepthStencil;
  D3DFORMAT AutoDepthStencilFormat;
  DWORD Flags;
  UINT FullScreen_RefreshRateInHz;
  UINT PresentationInterval;
} D3DPRESENT_PARAMETERS, *LPD3DPRESENT_PARAMETERS;
```

Struktura se na první pohled může zdát být složitá. Pro první program v Direct3D je však potřeba znát pouze dvě nastavení, jejichž význam je uveden níže. Více informací o nastavení struktury se lze dočíst zde [7].

- *hDeviceWindow* – Manipulátor okna aplikace.
- *Windowed* – Hodnota TRUE používá okna, hodnota FALSE pracuje v celo-obrazovkovém režimu.

Zbývající členské proměnné této struktury lze vyplnit hodnotami uvedenými ve výpisu 2.2.

Výpis 2.2 Nastavení struktury D3DPRESENT_PARAMETERS

```
D3DParametry     : TD3DPRESENTPARAMETERS;
. . .
ZeroMemory(@D3DParametry, SizeOf(D3DParametry));
with D3DParametry do begin
   SwapEffect         := D3DSWAPEFFECT_DISCARD;
   hDeviceWindow      := Handle;
   BackBufferCount    := 1;
   Windowed           := FALSE;
   BackBufferWidth    := 800;
   BackBufferHeight   := 600;
   BackBufferFormat   := D3DFMT_X8R8G8B8;
end;
```

Funkce *ZeroMerory()* inicializuje celou strukturu nulovou hodnotou. Proměnná *Windowed* je nastavena na FALSE a jako manipulátor proměnné *hDeciceWindow* je použit globální manipulátor Handle.

Posledním argumentem volání metody *CreateDevice()* je *ppReturnedDevice-Interface*. Ten přestavuje adresu ukazatele na rozhraní *IDirect3DDevice9*.

2.3 Režimy zobrazení a pixelové formáty

Režim zobrazení lze považovat za kombinaci rozlišení obrazovky a barevné hloubky. V současné době většina počítačů pracuje v rozlišení 1024x768 s 32-bitovými barvami. Můžeme však využívat libovolné jiné zobrazení, které podporuje grafické zařízení. Většina podporovaných režimů zobrazení je však zastaralá a pro programátora v Direct3D nemá význam.

Počítač zobrazuje body o konkrétní barvě, proto je potřeba znát něco o pixelovém formátu. Pixelový formát určuje, jak je informace o daném bodu uložena v paměti počítače. Počet bitů v barvě určuje, kolik paměti grafické karty každý pixel na obrazovce potřebuje.

Barvy jsou tvořeny kombinací červené, zelené a modré (RGB). Nejčastěji používaná hloubka barev je 32bitová, která pro každý pixel používá 4 bajty. První tři bajty představují množství dané barvy, čtvrtý určuje průhlednost barvy. Tomuto čtvrtému bajtu se říká alfa. V tabulce 2.1 jsou vidět některé pixelové formáty a konstanty, které jsou pro ně v SDK DirectX definovány.

Tabulka 2.1 Pixelové formáty

Formát	Popis
D3DFMT_X1R5G5B5	16bitový pixelový formát s 5 bity pro každou barvu
D3DFMT_A1R5G5B5	16bitový pixelový formát s 5 bity pro každou barvu a 1 bitem pro alfa
D3DFMT_A4R4G4B4	16bitový pixelový formát ARGB se 4 bity pro každou barvu a alfa
D3DFMT_X4R4G4B4	16bitový pixelový formát RGB se 4 bity pro každou barvu
D3DFMT_A8R8G8B8	32bitový pixelový formát ARGB s 8 bity pro každou barvu a alfa
D3DFMT_X8R8G8B8	32bitový pixelový formát ARGB s 8 bity pro každou barvu (čtvrtý byte je nevyužit)
D3DFMT_A2B10G10R10	32bitový pixelový formát s 10 bity pro každou barvu a 2 bity pro alfa
D3DFMT_G16R16	32bitový pixelový formát s 16 bity pro zelenou a červenou barvu

2.4 Vykreslení zobrazení

Pro vykreslení na obrazovku je třeba znát několik dalších metod Direct3D. První z nich je metoda *Clear()*, která představuje metodu objektu zařízení Direct3D. V SDK je tato metoda deklarována takto:

```
HRESULT Clear(
  DWORD Count,
  CONST D3DRECT * pRects,
  DWORD Flags,
  D3DCOLOR Color,
  float Z,
  DWORD Stencil
);
```

Tato metoda má tyto argumenty, jejichž význam je následující:

- *Count* – Počet obdélníků v poli pObdelniky; v případě, že je pObdelniky rovno NIL, je tato hodnota rovna nule.
- *pRect* – Ukazatel na pole struktur D3DRECT, které popisují obdélníky, jež mají být odstraněny; pokud je použita hodnota NIL, je odstraněno vše v zorném poli aplikace.

- *Flags* – Příznaky určující povrchy, které mají být odstraněny. Může se jednat o kombinaci D3DCLEAR_STENCIL, D3DCLEAR_TARGET, D3DCLEAR_ZBUFFER.
- *Color* – 32bitová hodnota určující barvu, kterou mají být obdélníky nahrazeny.
- *Z* – Nová hodnota Z pro hloubkový buffer.
- *Stencil* – Celočíselná hodnota uložená ve stencil bufferu.

Volání metody *Clear()* vypadá následovně:

```
g_pZarizeniDirect3D.Clear(0, nil, D3DCLEAR_TARGET,
  D3DCOLOR_XRGB(red,green,blue), 0, 0 );
```

Argument D3DCLEAR_TARGET říká Direct3D, aby vykreslilo pozadí aplikace barvou uvedenou v argumentu D3DCOLOR. Hodnotu tohoto argumentu lze vytvořit pomocí makra XRGB, které pracuje s intenzitou jednotlivých složek RGB. Zbývající argumenty mohou být nulové.

Po vyprázdnění se plocha musí zobrazit. To se provede zavoláním metody *Present()* objektu zařízení Direct3D. Deklarace metody v SDK DirectX je následující:

```
HRESULT Present(
  CONST RECT* pSourceRect,
  CONST RECT* pDestRect,
  HWND hDestWindowOverride,
  CONST RGNDATA* pDirtyRegoin
);
```

Pro účely programů, které jsou zde vytvořeny postačí následující volání metody *Present()*. Toto volání zobrazí dostupnou plochu a přepíše všechno, co na obrazovce bylo.

```
Present(nil, nil, 0, nil);
```

2.5 Uvolnění objektů Direct3D

Jelikož program alokujeme paměť, musí ji před ukončením aplikace také opět uvolnit. To platí i pro objekty Direct3D, proto se po ukončení práce s nimi musí odstranit z paměti. Znamená to, že jednotlivým objektům přiřadíme hodnotu NIL. V programu je vytvořená procedura *UvolněníD3D* jejíž obsah je v následujícím výpisu 2.3.

Výpis 2.3 Uvolnění objektů

```
if assigned(g_pZarizeniDirect3D) then g_pZarizeniDirect3D:=nil;
if assigned(g_pDirect3D) then g_pDirect3D:=nil;
```

Při práci s těmito objekty je potřeba dodržovat některé věci. Zaprvé je před uvolněním objektů potřeba zkontrolovat, zda nejsou jejich hodnoty rovny NIL.

Jestliže se podařilo úspěšně vytvořit objekt Direct3D, pak tato hodnota v žádném případě není rovna NIL. Druhou neméně důležitou věcí je způsob uvolňování objektů – program musí provést uvolnění v opačném pořadí, než v jakém byly objekty vytvořeny. Při deklaraci je objekt Direct3D vytvořen jako první, proto je uvolněn vždy jako poslední.

2.6 Program Vytvoření objektu Direct3D

V tomto demonstračním programu (viz výpis 2.4) je ukázáno vytvoření objektu Direct3D. Aplikace vytvoří okno v celoobrazovkovém zobrazení a vykreslí modré pozadí. Program lze ukončit stisknutím klávesy ESC.

Výpis 2.4 Vytvoření objektu Direct3D

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  g_pDirect3D          :=nil;
  g_pZarizeniDirect3D  :=nil;
end;
procedure TForm1.FormShow(Sender: TObject);
begin
  InicializaceD3D;
  VykresleniD3D;
end;
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if Key=VK_ESCAPE then close;
end;
procedure TForm1.FormDestroy(Sender: TObject);
begin
  UvolneniD3D;
end;
procedure TForm1.InicializaceD3D;
var
  hr            : HRESULT;
  D3DParametry  : TD3DPRESENTPARAMETERS;
begin
  g_pDirect3D:=Direct3DCreate9(D3D_SDK_VERSION);
  ZeroMemory(@D3DParametry, SizeOf(D3DParametry));
  with D3DParametry do begin
    SwapEffect       := D3DSWAPEFFECT_DISCARD;
    hDeviceWindow    := Handle;
    BackBufferCount  := 1;
    Windowed         := FALSE;
    BackBufferWidth  := 800;
    BackBufferHeight := 600;
    BackBufferFormat := D3DFMT_X8R8G8B8;
  end;
  hr:=g_pDirect3D.CreateDevice(
    D3DADAPTER_DEFAULT,
    D3DDEVTYPE_HAL,
    Handle,
    D3DCREATE_SOFTWARE_VERTEXPROCESSING,
    @D3DParametry,
```

```
      g_pZarizeniDirect3D);
    if FAILED(hr) then Exit;
  end;
  procedure TForm1.VykresleniD3D;
  begin
    if assigned(g_pZarizeniDirect3D) then
    with g_pZarizeniDirect3D do begin
      Clear(0, nil, D3DCLEAR_TARGET, D3DCOLOR_XRGB(0,0,255), 0, 0);
      Present(nil, nil, 0, nil);
    end;
  end;
  procedure TForm1.UvolneniD3D;
  begin
    if assigned(g_pZarizeniDirect3D) then
      g_pZarizeniDirect3D:=nil;
    if assigned(g_pDirect3D) then g_pDirect3D:=nil;
  end;
```

Prvním krokem při psaní programu v Direct3D je vytvoření objektu Direct3D, jenž nabízí přístup k rozhraní *IDirect3D9*. Po vytvoření objektu se musí vytvořit zařízení Direct3D, které představuje rozhraní *IDirect3DDevice9*. Pro vykreslení zobrazení se musí zavolat metoda *Present()* objektu zařízení Direct3D. Po dokončení práce s objekty Direct3D je potřeba je odstranit z paměti. Objekty se uvolňují v opačném pořadí, než se vytvářejí.

3 Povrchy v Direct3D

Povrchy v Direc3D jsou pro programátory velice důležité, jelikož umožňují aplikacím ukládat grafiku a pracovat s ní. Povrchy určují, co bude na obrazovce. V této části je ukázáno proč je tomu tak. Jmenovitě je zde věnována pozornost následujícím bodům:

- K čemu jsou povrchy v programu.
- Různé druhy povrchů.
- Vytváření povrchů a přístup k nim.

3.1 Co je to povrch

V této práci je často pracováno s něčím, co se nazývá povrch. Povrch je oblast paměti, do které můžeme ukládat grafické informace. Vše co vidíme na obrazovce nebo v okně (jednotlivé obrázky či obrázky, ze kterých se obrazovka skládá) je uloženo v povrchu. Zde se dozvíme o čtyřech různých typech povrchů:

- Primární povrch.
- Povrch zadního bufferu.
- Obrazové povrchy.
- Texturové povrchy.

3.1.1 Primární povrch

Primární povrch, často též nazývaný přední buffer,je to, co vidíme při spuštění aplikace v Direct3D na obrazovce. Primární povrch obsahuje grafická data celé obrazovky. Primární povrch je jediným typem povrchu, který musí všechny programy v Direct3D vytvořit a spravovat. Vytvoření objektu zařízení Direct3D má automatiky za následek i vytvoření primárního povrchu. O explicitní vytváření tohoto povrchu se tedy nemusíme starat.

3.1.2 Povrch zadního bufferu

Zadní buffer je oblast paměti, kde je možné vykreslit scénu mimo obrazovku. Po tomto vykreslení je zavolána metoda *Present()*, která přenese vykreslenou obrazovku ze zadního bufferu do primárního povrchu. Používáním zadního bufferu je zamezeno věcem jako je blikání obrazu. Zadní buffer je důležitý pro vykreslení animací. Lze jej

automaticky vytvořit při zakládání objektu zařízení Direct3D ve struktuře D3DPRESENT_PARAMETRS. Struktura je definována v předchozí kapitole. Zde je jen doplněn význam proměnných týkající se zadního bufferu:

- *BackBufferWidth* – Šířka zadního bufferu.
- *BackBufferHeight* – Výška zadního bufferu.
- *BackBufferFormat* – Pixelový formát zadního bufferu.
- *BackBufferCount* – Počet zadních bufferů, které mají být vytvořeny.

3.1.3 Obrazové povrchy

Aplikace v Direct3D potřebují prostor pro ukládání obrázků, se kterými program pracuje. Představme si, že chceme použít bitmapu jako scénu na pozadí. Nejprve je zapotřebí načíst bitmapu do paměti, tedy do obrazového povrchu. Obrazový povrch musí být v programu explicitně vytvořen. To provádí metoda objektu zařízení Direct3D nazvaná *CreateOffscreanPlainSurface()*, která je v SDK DirectX deklarována takto:

```
HRESULT CreateOffscreenPlainSurface(
  UINT Width,
  UINT Height,
  D3DFORMAT Format,
  DWORD Pool,
  IDirect3DSurface9** ppSurface,
  HANDLE* pSharedHandle,
);
```

Význam jednotlivých argumentů je následující:

- *Width* – Požadovaná šířka povrchu.
- *Height* – Požadovaná výška povrchu.
- *Format* – Pixelový formát povrchu.
- *Pool* – Pool povrchu, pro účel této práce je použita hodnota D3DPOOL_DEFAULT.
- *PpSurface* – Adresa, na kterou má metoda uložit ukazatel na nový objekt povrchu.
- *pHandle* – Pro účel práce je použita hodnota NIL.

V ukázkové aplikaci vypadá načtení do obrázkového bufferu následovně:

Výpis 3.1 Vytvoření povrchu

```
g_pObrazovyPovrch    : IDirect3DSurface;
g_hr                 : HRESULT;
. . .
g_pObrazovyPovrch := nil;
g_hr:=g_pZarizeniDirect3D.CreateOffscreenPlainSurface(640, 480,
  D3DFMT_X8R8G8B8, D3DPOOL_DEFAULT, g_pObrazovyPovrch, nil);
if FAILED(g_hr) then Exit;
```

Když je povrch vytvořen je potřeba do něj bitmapu načíst. SDK DirectX poskytuje pro tento účel funkci *D3DCLoadSurfaceFromFile()*, která je deklarována následovně:

```
HRESULT D3DXLoadSurfaceFromFile(
  LPDIRECT3DSURFACE9 pDestSurface,
  CONST PALETTEENTRY* pDestPalette,
  CONST RECT* pDestRect,
  LPCTSTR pSrcFile,
  CONST RECT* pSrcRect,
  DWORD Filter,
  D3DCOLOR ColorKey,
  D3DXIMAGE_INFO* pSrcInfo
);
```

Jednotlivé argumenty maní následující význam:

- *pDestSurface* – Ukazatel na povrch, do kterého má být obrázek načten.
- *pDestPalette* – Ukazatel na paletu, do které mají být načteny barvy obrázku.
- *pDestRect* – Obdélníková oblast povrchu, do kterého má být obrázek načten; hodnota NIL označuje celý dostupný povrch.
- *pSrcFile* – Cesta k souboru, jenž má být načten.
- *Filter* – Typ filtru, který má být použit, zpravidla se ale jedná o D3DX_DEFAULT.
- *ColorKey* – Barevná hodnota, která by měla být změněna na průhlednou černou nebo na 0, pokud má být barevné klíčování vypnuté.
- *pSrcInfo* – Ukazatel na strukturu D3DXIMAGE_INFO obsahující informace o obrázku. povrchu.

Ve výpisu 3.2 je ukázáno načítání obrázku do povrchu:

Výpis 3.2 Načtení obrázku ze souboru

```
g_hr:=D3DXLoadSurfaceFromFile(g_pObrazovyPovrch, nil, nil,
  'povrch.jpg', nil, D3DX_DEFAULT, 0, nil);
if FAILED(g_hr) then Exit;
```

3.2 Přenos obrázků mezi povrchy

Nyní je vytvořen obrázkový povrch a do něj je načten obsah souboru z disku. Zbývá obrázek pouze zobrazit. K tomu je zapotřebí přenést obrázek do zadního bufferu a poté zavolat metodu objektu zařízení *Present()*.

Nejprve je potřeba se k zadnímu bufferu dostat. Požadovaný ukazatel se získá zavoláním metody objektu zařízení Direct3D *GetBackBuffer()*, která je v SDK definovaná následovně:

```
HRESULT GetBackBuffer(
  UINT  iSwapChain,
  UINT BackBuffer,
  D3DBACKBUFFER_TYPE Type,
  IDirect3DSurface9 ** ppBackBuffer
);
```

Metoda má následující čtyři argumenty:

- *iSwapChain* – Identifikátor swapovacího řetězu.
- *BackBuffer* – Index požadovaného zadního bufferu. Pro použití jednoho zadního bufferu, je tato hodnota 0.
- *Type* – Hodnota parametru musí být D3DBACKBUFFER_TYPE_MONO.
- *ppBackBuffer* – Adresa, kam má metoda uložit vrácený ukazatel.

Příklad volání metody *GetBackBuffer()* v ukázkovém programu je uveden ve výpisu 3.3.

Výpis 3.3 Získání ukazatele na zadní buffer

```
pZadniBuffer  : IDirect3DSurface9;
. . .
pZadniBuffer  := nil;
g_hr:=g_pZarizeniDirect3D.GetBackBuffer(0,0,
  D3DBACKBUFFER_TYPE_MONO, pZadniBuffer);
if FAILED(g_hr) then Exit;
```

Po získání ukazatele na povrch zadního bufferu je vše připraveno na volání metody *StretchRect()*, kterou SDK DirectX definuje takto:

```
HRESULT StretchRect(
  IDirect3DSurface9 * pSourceSurface,
  CONST RECT * pSourceRect,
  IDirect3DSurface9 * pDestSurface,
  CONST RECT * pDestRect,
  D3DTEXTUREFILTERTYPE Filter
);
```

Metoda pracuje s těmito argumenty:

- *pSourceSurface* – Ukazatel na povrch, ze kterého mají být data kopírována.
- *pSourceRect* – Ukazatel na strukturu RECT obsahující souřadnice obdélníku, který se má kopírovat. Pro zkopírování celého povrchu se použije hodnota NIL.
- *pDestSurface* – Ukazatel na povrch, do kterého mají být data kopírována.
- *pDestRect* – Ukazatel na strukturu RECT obsahující souřadnice cílového obdélníku.
- *Filter* – Typ filtru, který může být D3DTESTF_NONE, D3DTESTF_POINT, D3DTESTF_LINEAR.

Kopírování bitmapy mezi povrchy je uvedeno ve výpisu 3.4.

Výpis 3.4 Kopírování bitmapy mezi povrchy

```
g_hr:=g_pZarizeniDirect3D.StretchRect(g_pObrazovyPovrch, nil,
  pZadniBuffer, nil, D3DTEXF_NONE);
if FAILED(g_hr) then Exit;
```

Nyní je třeba zavolat metodu objektu zařízení Direct3D *Present()* a zobrazit načtenou bitmapu. V následující podkapitole je popsán program demonstrující použití povrchů v Direct3D.

3.3 Program Povrchy

Program vytváří a načítá povrchy v Direct3D a následně mezi nimi přenáší data obrázku. Po načtení dat obrázku aplikace bitmapu zobrazí. Prakticky všechny programy v Direct3D používají tyto techniky, proto patří v Direct3D k nejdůležitějším. Program povrchy lze shlédnout na přiloženém CD.

Do kořenového adresáře nesmí být zapomenuta nakopírována daná bitmapa, která je vykreslena jako pozadí, popřípadě je třeba uvést v metodě *D3DXLoadSurface-FromFile()* cestu k této bitmapě. Program vykreslí bitmapu na pozadí scény a jako u předchozí aplikace se ukončí klávesou ESC.

4 Vykreslení grafických primitiv

Součástí práce s grafickou knihovnou je i vykreslování základních grafických primitiv, jako jsou body, čáry nebo obdeníky. Knihovna Direct3D pro vykreslení těchto tvarů pracuje s body a vrcholy. V této kapitole je ukázáno, jak Direct3D využívá právě tyto body a vrcholy k vykreslování grafických primitiv. Postupně se věnuje:

- Definování tvarů pomocí vrcholů.
- Vytvoření a používání vertex bufferu.
- Používání Direct3D k vykresleni základních tvarů.

4.1 Definování tvarů pomocí vrcholů

Kreslení objektů s používáním bodů by bylo samo o sobě velice složité. Proto programátoři používají pro vykreslení složitějších tvarů vrcholů (vertexů). Vrchol je totiž bodem, který určuje, kde se spojují dvě hrany útvaru. Jedná se o zvláštní typ bodu, který je definován podobně jako obyčejný bod, tedy pomocí souřadnic X, Y a Z. Pokud chceme na obrazovku vykreslit trojúhelník, musíme mu definovat tři vrcholy, jak je vidět na obrázku 4.1. nebo na obrázku 4.2.

4.2 Vertex buffer

Vertex buffer je rezervované místo v paměti, kde jsou uloženy informace o vrcholech potřebné k vykreslení požadovaného tvaru. Jako příklad je zde uvedeno vykresleni trojúhelníku. Pro tento případ je tedy potřeba vytvořit vertex buffer, který uchovává tři vrcholy trojúhelníku. Před vytvořením bufferu, musí Direct3D znát vlastnosti těchto vrcholů. Těmito vlastnostmi se kromě souřadnic rozumí třeba barva, souřadnice textury, informace o míchání apod. V následujícím výpisu 4.1 je ukázáno jak lze tento datový typ deklarovat. Vrchol obsahuje pouze souřadnice a barvu. Direct3D používá k popisu vrcholů příznaky, které jsou postupně dále popsány. Zde je vytvořen nový datový typ *VLASTNIVERTEX*, který je použit k definici trojúhelníku.

Výpis 4.1 Deklarace typu vrcholu

```
type
VLASTNIVERTEX = record
  x, y, z, rhw  : Single;
  barva         : LongWord;
end;
```

K vytvoření sady vrcholů, které určují tvar výsledného trojúhelníku je použito následujícího zápisu:

Výpis 4.2 Definice vrcholů trojúhelníku

```
const
VrcholyTrojuhelniku: array [0..2] of VLASTNIVERTEX = (
   (x: 400.0; y: 180.0; z: 0.0; rhw: 1.0; barva: $FF0000),
   (x: 500.0; y: 380.0; z: 0.0; rhw: 1.0; barva: $00FF00),
   (x: 300.0; y: 380.0; z: 0.0; rhw: 1.0; barva: $0000FF));
D3DFVF_CUSTOMVERTEX = (D3DFVF_XYZRHW or D3DFVF_DIFFUSE);
```

Deklarací této konstanty je nadefinován trojúhelník, který se objeví uprostřed obrazovky, při použití rozlišení 800x600. Vrcholy představují obrazovkové souřadnice s počátkem souřadného systému v levém horním rohu obrazovky a s kladnými hodnotami X směrem doprava a kladnými hodnotami Y směrem dolů.

Nejjednodušší typ vrcholu určuje příznak D3DFVF_XYZ, který popisuje netransformované souřadnice. Proto je potřeba definovat typ jiný. Vhodným typem vrcholu je D3DFVF_XYZRHW, který říká Direct3D, že vrchol je definován souřadnicemi a hodnotou RHW. V našem případě je rovna hodnota tohoto parametru 1.0. Posledním parametrem vrcholu, který je definován je barva. Barva je dána hodnotami RGB jednotlivých barvených složek.

Druhá konstanta D3DFVF_CUSTOMVERTEX je deklarována pro usnadnění další práce s vertex bufferem.Obsahuje kromě již zmíněného D3DFVF_XYZRHW příznaku příznak D3DFVF_DIFFUSE, který říká Direct3D, že je použito barevné hodnoty spolu s hodnotami X, Y, Z a RHW.

4.2.1 Vytvoření vertex bufferu

Nyní je nadefinován datový typ pro vrchol a pole vrcholů trojúhelníku. Dále je potřeba vytvořit vertex buffer, do které jsou hodnoty vrcholů ukládány. Pro tento úkol nabízí objekt zařízení Direct3D metodu *CreateVertexBuffer()*, která je v SDK DirectX definována následovně:

```
HRESULT CreateVertexBuffer(
   UINT Length,
   DWORD Usage,
   DWORD FVF,
   D3DPOOL Pool,
   IDirect3DVertexBuffer9** ppVertexBuffer,
   HANDLE* pSharedHandle
);
```

Význam argumentů metody *CreateVertexBuffer()* je následující:

- *Length* – Požadovaná velikost bufferu.
- *Usage* – Příznaky určující způsob použití vertex bufferu.
- *FVF* – Formát vrcholu. Zde je použita konstanta D3DFVF_CUSTOMVERTEX.
- *Pool* – Hodnota určující paměťové využití bufferu.
- *ppVertexBuffer* – Adresa, kde má metoda ukládat ukazatele na nově vytvořený objekt vertex bufferu.
- *pSharedHandle* – V diplomové práci je použita hodnota NIL.

V aplikaci vykreslující trojúhelník vypadá volání metody pro vytvoření vertex bufferu následovně:

Výpis 4.3 Vytvoření vertex bufferu

```
g_pVertexBuffer  : IDirect3DVertexBuffer9;
hr               : HRESULT;
. . .
g_pVertexBuffer := nil;
hr:= g_pZarizeniDirect3D.CreateVertexBuffer(
  3*SizeOf(VLASTNIVERTEX), 0,
  D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT,
  g_pVertexBuffer, nil);
if FAILED(hr) then Exit;
```

4.2.2 Načtení vertex bufferu

Závěrečný úkon při vytváření vertex bufferu je načtení vrcholů do bufferu. Než tak lze učinit, musí být buffer zamčen, aby Direct3D nepracovalo s jeho daty během jeho práce. Buffer se zamyká zavoláním metody vertex bufferu *Lock()*, která je v Direct3D deklarována následovně:

```
HRESULT Lock(
  UINT OffsetToLock,
  UINT SizeToLock,
  VOID ** ppbData,
  DWORD Flags
);
```

Význam argumentů funkce *Lock()* je následující:

- *OffsetToLock* – Index prvního zamčeného bajtu.
- *SizeToLock* – Počet bajtů, které mají být zamčeny. Hodnota 0 zamkne celý buffer.
- *ppbData* – Adresa, kam má metoda uložit ukazatel na data ve vertex bufferu.
- *Flags* – Sada příznaků, které definují zamknutí bufferu.

Kompletní načtení vertex bufferu je ukázáno ve výpisu 4.4. Nejprve je buffer zamčen, poté se přenáší data z programu do bufferu a následně je vertex buffer opět odemčen.

Výpis 4.4 Načtení vertex bufferu

```
hr:= g_pVertexBuffer.Lock(0,SizeOf(VrcholyTrojuhelniku),pVrcholy,0);
if FAILED(hr) then Exit;
CopyMemory(pVrcholy, @VrcholyTrojuhelniku,
  SizeOf(vrcholyTrojuhelniku));
g_pVertexBuffer.Unlock;
```

4.3 Základních primitiva

Ke kreslení tvarů (polygonů) používá Direct3D šest základních typů grafických primitiv v sadách zvaných seznamy (proužky). Pomocí těchto základních grafických primitiv lze v Direct3D vykreslit jakýkoliv objekt. Tyto základní primitiva jsou následující:

- *Seznamy bodů* – Seznam samostatný bodů.
- *Seznamy čar* – Seznam samotných čar.
- *Proužky čar* – Seznam spojených čar.
- *Proužky trojúhelníků* – Seznam spojených trojúhelníků.
- *Vějíře* – Seznam trojúhelníků spojených do tvaru vějíře.

Aby Direct3D mohlo vykreslit tvar na obrazovku, musí se mu dodat několik informací. Direct3D musí vědět, kde jsou vrcholy uloženy, jak mají být tvary stínovány a jaký typ tvaru má být vykreslen. K určení umístění vrcholů se používá metoda objektu zařízení Direct3D *SetStreamSource()*, která je v SDK definována takto:

```
HRESULT SetStreamSource(
  UINT StreamNumber,
  IDirect3DVertexBuffer9 * pStreamData,
  UINT OffsetInBytes,
  UINT Stride
);
```

Význam čtyř argumentů funkce *SetStreamSource()* je následující:

- *StreamNumber* – Použitý datový tok. Ve většině případů je hodnota rovna 0.
- *sStreamData* – Ukazatel na objekt vertex bufferu, jenž obsahuje vrcholy k vykreslení.
- *OffsetInBytes* – Počet bajtů od začátku bufferu k datům vrcholů.
- *Stride* – Velikost každé sady s daty vrcholů.

Následují instrukce říká Direct3D, aby aplikoval na grafická primitiva stínování. Pro tento účel nabízí Direct3D metodu objektu zařízení *SetFVF()*, jejíž deklarace v SDK vypadá následovně:

```
HRESULT SetFVF(
  DWORD FVF
);
```

Jediným argumentem této metody je FVF a představuje hodnotu, která uchovává příznaky o formátu vrcholu.

Posledním krokem k vykreslení tvaru, je zavolání metody objektu zařízení Direct3D *DrawPrimitive()*, která je v Direct3D deklarována následovně:

```
HRESULT DrawPrimitive(
  D3DPRIMITIVETYPE PrimitiveType,
  UINT StartVertex,
  UINT PrimitiveCount
);
```

Tři argumenty metody *DrawPrimitive()* mají následující význam:

- *PrimitiveType* – Jedna z konstant, které určují typ primitiva.
- *StartVertex* – Vrchol, ve kterém začíná program kreslit.
- *PrimitiveCount* – Počet tvarů, které mají být vykresleny.

Nyní je vše připraveno k vykreslení tvaru na obrazovku. Zbývá však dodat, že než se začne na obrazovku kreslit, musí se zavolat metoda objektu zařízení Direct3D *BeginScene()* a po dokončení kreslení metoda *EndScene()*. Pro úplnou přesnost musí být ještě zavolána metoda *Present()*. Názorně je to ukázáno ve výpisu 4.5.

Výpis 4.5 Vykreslení tvaru na obrazovku

```
SetStreamSource(0, g_pVertexBuffer, 0, SizeOf(VLASTNIVERTEX));
SetFVF(D3DFVF_CUSTOMVERTEX);
BeginScene;
  DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
EndScene;
Present(nil, nil, 0, nil);
```

4.4 Aplikace Trojúhelník

Ukázková aplikace nazvaná *Trojúhelník* demonstruje techniky programování, jež jsou uvedeny výše. Program vykreslí trojúhelník doprostřed obrazovky. Každý vrchol trojúhelníku má jinou barvu. Diky použití typu vrcholu D3DFVF_DIFFUSE se barva mezi vrcholy míchá. Aplikace *Trojúhelník* je na přiloženém CD. V programu přibyla oproti předchozí aplikaci, ze které vychází všechna další práce, procedura *InicializaceSceny*, ve které je vytvořen vertex buffer, který obsahuje informace o vrcholech trojúhelníku.

4.5 Kreslení základních primitiv

Jak již bylo zmíněno výše, pro kreslení jakéhokoliv tvaru v Direct3D se používá základních primitiv (seznamů bodů, čar, proužků čar, trojúhelníků a vějířů). V následující části je popsáno vykreslení těchto primitiv.

4.5.1 Vykreslení seznamu bodů

Nejjednodušším tvarem, který lze na obrazovku vykreslit je bod. Bod není nic jiného než samostatná tečka na obrazovce. V programu vykreslujícím základní grafická primitiva je vykreslena série bodů, jejichž definice probíhá v seznamech s použitím konstanty D3DPT_POINTLIST. Tato definice je velice podobná definování vrcholů trojúhelníku.

4.5.2 Vykreslení seznamu čar

Dalším typem tvarů, které lze v Direct3D vykreslit, je čára. Podobně jako v případě bodů lze vykreslit buď jednu, nebo více čar. Ty se definují jako seznamy čar s použitím konstanty D3DPT_LINELIST. Formát čáry je stejný jako u definování bodů, jelikož čáry jsou definovány počátečním a koncový bodem.

4.5.3 Vykreslení proužků čar

Dalším jednoduchým tvarem, který dokáže Direct3D vykreslit, je proužek čar. Ten lze definovat s použitím konstanty D3DPT_LINESTRIP. Seznam představuje množinu propojených čar, z nichž je první čára definována prvníma dvěma body a každá další koncovým bodem předchozí čáry a vlastním koncovým bodem.

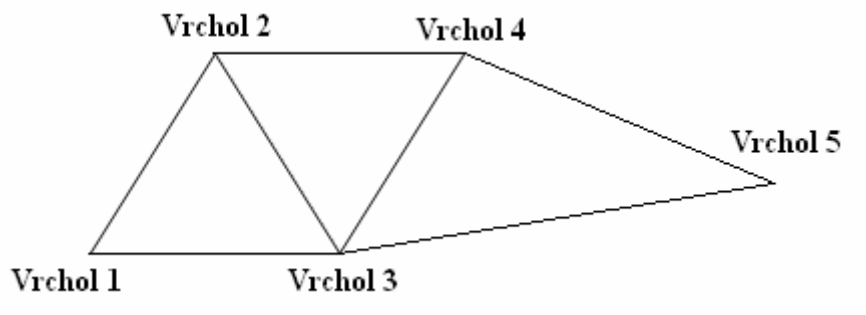
4.5.4 Vykreslení seznamu trojúhelníků

Vykreslení seznamu trojúhelníků je ukázáno na začátku této kapitoly a v ukázkové aplikaci *Trojúhelník*. Vykreslen je však pouze jeden trojúhelník. Seznam trojúhelníků se definuje s použitím konstanty D3DPT_TRIANGLELIST.

4.5.5 Vykreslení proužku trojúhelníků

Dalším grafickým primitivem, který lze v Direct3D vykreslit, je proužek trojúhelníků, jenž představuje sérii spojených trojúhelníků. První trojúhelník je definován třemi vrcholy. Každý následující trojúhelník sdílí s předchozím jednu hranu

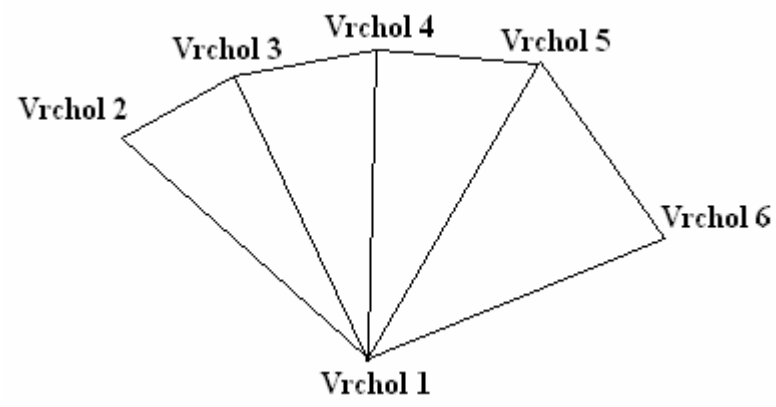
a je definován pomocí jednoho vrcholu, jak je ukázáno na obrázku 4.1. Pořadí, ve kterém jsou vrcholy proužku trojúhelníku definovány, je velice důležité. Seznam trojúhelníků definujeme konstantou D3DPT_TRIANGLESTRIP.



Obrázek 4.1 Definování vrcholů seznamu trojúhelníků

4.5.6 Vykreslení vějíře

Posledním základním primitivem v Direct3D jsou vějíře, které jsou definovány seznamem trojúhelníku s použitím konstanty D3DPT_TRIAGLEFAN. Vějíř představuje sérii trojúhelníků, jež sdílí jediný vrchol, který představuje střed vějíře. Definice takového vějíře je naznačena na obrázku 4.2.



Obrázek 4.2 Definování vrcholů vějíře

Programy demonstrující vykreslení základních primitiv lze nalézt v příslušném adresáři na přiloženém CD. V aplikacích je nejdříve deklarován typ vrcholu, následně je definována množina bodů (vrcholů) a na závěr je v proceduře *VykreslenìD3D* v metodě objektu zařízení Direct3D *DrawPrimitive()* definována konstanta určující typ vykresleného primitiva.

5 Transformace

5.1 Transformace v Direct3D

Základní typy transformací v trojrozměrném prostoru jsou posun, rotace a změna velikosti. Direct3D nabízí sadu transformací na vyšší úrovni, bez kterých se programátor neobejde při vykreslování trojrozměrného virtuálního světa. Těmito transformacemi jsou transformace objektu, pohledová a projekční transformace. V této kapitole je ukázáno, co které transformace znamenají.

5.1.1 Transformace objektu

Transformace objektu jsou vlastně základní transformace jako posun, rotace či změna velikosti. Transformace určuje umístění a pozici grafických objektů ve scéně.

Direct3D používá pro provádění transformací objektů matice. Deklarací matice začíná transformace objektu. Pro usnadnění práce s těmito maticemi nabízí Direct3D datový typ *TD3DXMATRIX*.

Knihovna Direct3D nabízí několik funkcí pro transformace. Direct3D stačí pouze říct, jak chceme posun, rotaci či změnu velikosti provádět. Tyto funkce jsou následující:

- *D3DXMatrixTranslation()* – Posunová matice.
- *D3DXMatrixRotationX()* – Rotační matice pro osu X.
- *D3DXMatrixRotationY()* – Rotační matice pro osu Y.
- *D3DXMatrixRotationZ()* – Rotační matice pro osu Z.
- *D3DXMatrixScaling()* – Matice pro změnu velikosti.

Po nastavení matice pro transformaci se musí výsledná transformace, která se může skládat i z více transformací, předat Direct3D zavoláním metody zařízení Direct3D *SetTransform()*. Tuto metodu Direct3D deklaruje následovně:

```
HRESULT SetTransform(
  D3DTRANSFORMSTATETYPE State,
  CONST D3DMATRIX * pMatrix
);
```

Argumenty mají následující význam:

- *State* – Typ použité transformace.
- *pMatrix* – Ukazatel na matici transformace.

5.1.1.1 Posun

Pro umístění objektů ve virtuálním světě, je třeba provést posun ve směru osy X, Y, Z nebo jejich kombinací. K těmto účelům nabízí Direct3D metodu *D3DXMatrix-Translation()*, kterou definuje takto:

```
D3DXMATRIX *D3DXMatrixTranslation(
  D3DXMATRIX * pOut,
  FLOAT x,
  FLOAT y,
  FLOAT z
);
```

A zde je význam argumentů funkce *D3DXMatrixTranslation()*:

- *pOut* – Ukazatel na matici uchovávající data transformace.
- *x* – Počet jednotek, o které má být posunut objekt ve směru osy X.
- *y* – Počet jednotek, o které má být posunut objekt ve směru osy Y.
- *z* – Počet jednotek, o které má být posunut objekt ve směru osy Z.

5.1.1.2 Rotace

Virtuální svět je tvořen třemi osami rotace. Proto Direct3D nabízí tři pomocné funkce pro rotaci: *D3DXMatrixRotationX() D3DXMatrixRotationY() D3DXMatrix-RotationZ().* Tyto funkce jsou v SDK Direct3D definovány následovně:

```
D3DXMATRIX *D3DMatrixRotationX(
  D3DXMATRIX * pOut,
  FLOAT Angle
);
```

Význam uvedených argumentů je následující:

- *pOut* – Ukazatel na matici uchovávající data transformace.
- *Angle* – Úhel natočení v radiánech.

Funkce *D3DXMatrixRotationY(),D3DXMatrixRotationZ()* fungují stejným způsobem, pouze rotují kolem své osy.

5.1.1.3 Změna velikosti

Jestliže potřebujeme zvětši nebo zmenšit objekt v trojrozměrné scéně, musí se zavolat metoda *D3DXMatrixScaling()*, která je definována následovně:

```
D3DXMATRIX *D3DMatrixScaling(
  D3DXMATRIX * pOut,
  FLOAT sx,
  FLOAT sy,
  FLOAT sz
);
```

Argumenty funkce *D3DXMatrixScaling()* mají tento význam:

- *pOut* – Ukazatel na matici uchovávající data transformace.
- *sx* – Faktor změny velikosti na ose X.
- *sy* – Faktor změny velikosti na ose Y.
- *sz* – Faktor změny velikosti na ose Z.

5.1.2 Pohledová transformace

Před prováděním pohledové transformace v Direct3D je potřeba definovat tři vektory datového typu *TD3DVECTOR*. První vektor představuje pozici oka, neboli místo, ze kterého je na scénu nahlíženo. Lze ho považovat také za pozici kamery. Druhý vektor určuje směr pohledu, tedy místo na které kamera ukazuje. A poslední vektor se nazývá horní vektor. Tento vektor určuje náklon kamery. Ve výpisu 5.1 aplikace vykreslující trojúhelník ve 3D je použito toto nastavení pohledové transformace, které definuje pozici kamery pět dílků ve směru osy Z. Záporné hodnoty u osy Z mají za následek oddálení kamery dále od scény. Náklon kamery je shodný s osou Y.

Výpis 5.1 Definování vektorů pohledové transformace

```
vPozice, vSmer, vNaklon :TD3DVector;
. . .
vPozice  :=D3DXVECTOR3(0.0, 0.0, -5.0);
vSmer    :=D3DXVECTOR3(0.0, 0.0, 0.0);
vNaklon  :=D3DXVECTOR3(0.0, 1.0, 0.0);
```

Po definování těchto vektorů je potřeba sestavit matici pomocí funkce *D3DMatrixLookAtLH()*, jejíž deklarace v SDK DirectX je následující:

```
D3DXMATRIX *D3DMatrixLookAtLH(
  D3DXMATRIX * pOut,
  CONST D3DXVECTOR3 * pEye,
  CONST D3DXVECTOR3 * pAt,
  CONST D3DXVECTOR3 * pUp
);
```

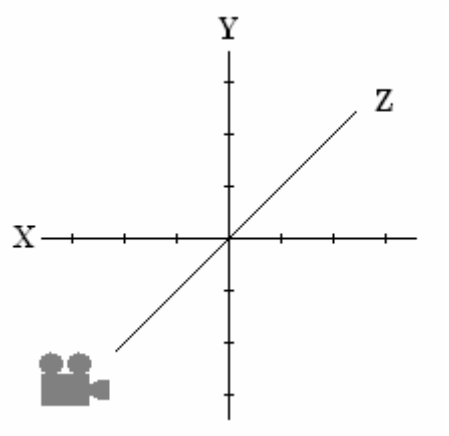
Uvedené argumenty mají tento význam:

- *pOut* – Ukazatel na matici uchovávající dat a transformace.
- *pEye* – Ukazatel na strukturu D3DXVECTOR3 určující pozici kamery ve scéně.
- *pAt* – Ukazatel na strukturu D3DXVECTOR3 určující směr pohledu kamery.
- *pUp* – Ukazatel na strukturu D3DXVECTOR3 určující náklon kamery.

V následujícím výpisu 5.2 je ukázáno volání metody *D3DXMatrixLookAtLH()* i s předchozím definováním vektorů určujících polohu, směr a náklon kamery ve virtuálním světě. Umístění kamery ve scéně si lze prohlédnout na obrázku 5.1. Poslední instrukce předává transformace do Direct3D pomocí metody *SetTranform()*.

Výpis 5.2 Nastavení pohledové transformace

```
vPozice, vSmer, vNaklon   : TD3DVector;
kamera        : TD3DMatrix;
. . .
vPozice  :=D3DXVECTOR3(0.0, 0.0, -5.0);
vSmer    :=D3DXVECTOR3(0.0, 0.0, 0.0);
vNaklon  :=D3DXVECTOR3(0.0, 1.0, 0.0);
D3DXMatrixLookAtLH(kamera, vPozice, vSmer, vNaklon);
g_pZarizeniDirect3D.SetTransform(D3DTS_VIEW, kamera);
```



Obrázek 5.1 Umístění kamery ve scéně

5.1.3 Projekční transformace

Když je scéna vytvořená a plně transformovaná, přichází na řadu problém, jak tuto trojrozměrnou virtuální scénu převést do dvourozměrného zobrazení, jelikož ve většině případů budeme tento virtuální svět chtít vidět na monitoru počítače. V Direct3D toto řeší projekční transformace, pro kterou je nejprve potřeba vytvořit matici. K tomu se používá pomocná funkce *D3DXMatrixPerspectiveFovLH()*, která je v DirectX definovaná následovně:

```
D3DXMATRIX *D3DXMatrixPerspectiveFovLH(
  D3DXMATRIX * pOut,
  FLOAT fovy,
  FLOAT Aspect,
  FLOAT zn,
  FLOAT zf
);
```

Význam jednotlivých argumentů je následující:

- *pOut* – Ukazatel na matici uchovávající data transformace.
- *fovY* – Úhel pohledu v radiánech.
- *Aspect* – Poměr šířky obrazu scény k jeho výšce.
- *zn* – Pozice blízké roviny ořezu scény.
- *zf* – Pozice vzdálené roviny ořezu scény.

V této práci je v ukázkových programech pracováno s nastavením projekční transformace ukázaném ve výpisu 5.3. Nastavení úhlu transformace je π/4, poměr šířky a výšky obrazu je 1, pozice blízké roviny ořezu scény je 1 a pozice vzdálené roviny ořezu scény je 50. Blízká rovina ořezu představuje vzdálenost od oka, odkud má být scéna vykreslena. Vzdálená rovina ořezu představuje vzdálenost od oka, po kterou má být scéna vykreslena. Jakékoliv objekty nebo jejich části před respektive za těmito rovinami ořezu se ve scéně neobjeví. I zde je potřeba výslednou matici předat Direct3D pomocí metody *SetTransform()*.

Výpis 5.3 Nastavení projekční transformace

```
projekce: TD3DMatrix;
. . .
D3DXMatrixPerspectiveFovLH(projekce, D3DX_PI/4, 1.0, 1.0, 50.0);
g_pZarizeniDirect3D.SetTransform(D3DTS_PROJECTION, projekce);
```

5.2 Program Trojúhelník ve 3D

Program vykresluje trojúhelník, který má vlastnosti trojrozměrného objektu. Na pozadí scény je nanesena bitmapa a v popředí scény je vykreslen trojúhelník. Největší rozdíl mezi vykreslením trojúhelníku ve 3D a předchozími programy spočívá ve formátu vrcholů, který je nyní D3DFVF_XYZ **or** D3DFVF_ DIFFUSE. Tento formát znamená, že data obsahují netransformované souřadnice s údajem o barvě. Netransformované souřadnice se stanou transformovanými v okamžiku provedení transformací objektu, pohledu a projekce.

Procedura *InicializaceD3D* obsahuje metodu *SetRenderState()* s argumenty D3DRS_LIGHTING, FALSE, které vypínají funkci Direct3D pro nasvícení. Dále v aplikaci přibyla procedura *InicializaceMatic*, kde se provádí nastavení transformace scény, pohledové a projekční transformace. Všechny tyto změny lze shlédnou v kompletním zdrojovém kódu aplikace *Trojúhelník ve 3D*, která je na CD přiloženém k diplomové práci.

5.3 Použití transformací v aplikaci Trojúhelník ve 3D

Použití transformací objektů je popsáno výše. Zde je ukázáno, jak tyto transformace fungují v praxi. Chceme-li trojúhelník posunout o jednu jednotku ve směru osy X a o dvě jednotku ve směru Y, učiníme tak následovně:

Výpis 5.4 Posunutí objektu

```
objekt : TD3DMatrix;
. . .
D3DXMatrixTranslation(objekt, 1, 2, 0);
g_pZarizeniDirect3D.SetTransform(D3DTS_WORLD, objekt);
```

Chceme-li objekt natočit o 45 stupňů kolem osy X, použijeme zápis uvedený ve výpisu 5.5. Rotace objektu kolem osy Y nebo Z má podobný zápis, jediná změna je ve funkci, kterou voláme. Pro rotaci kolem osy Y *D3DXMatrixRotationY()*, kolem osy Z *D3DXMatrix RotationZ()*.

Výpis 5.5 Rotace objektu

```
Objekt : TD3DMatrix;
radiany: Single;
. . .
radiany:= 6.2831 / (360.0 / 45);
D3DXMatrixRotationZ(objekt, radiany);
g_pZarizeniDirect3D.SetTransform(D3DTS_WORLD, objekt);
```

Pro zněnu velikosti objektu se používá metoda *D3DXMatrixScaling()*. Budeme-li chtít změnit velikost objektu na dvojnásobek, použijeme následující výpis 5.6.

Výpis 5.6 Změna velikosti objektu

```
objekt : TD3DMatrix;
. . .
D3DXMatrixScaling(objekt, 2, 2, 2);
g_pZarizeniDirect3D.SetTransform(D3DTS_WORLD, objekt);
```

5.4 Vytvoření krychle z polygonů

Ačkoliv se s trojúhelníkem pracuje ve virtuálním prostoru, nejedná se skutečně o trojrozměrný objekt. Pro dojem třetího rozměru je zde sestavena krychle skládající se z více trojúhelníků. Tato krychle je použita jako vzorová i pro následující aplikace. Zkrácená definice takové krychle je ukázána ve výpisu 5.7.

Výpis 5.7 Definování vrcholů krychle

```
type
VLASTNIVERTEX = record
  x, y, z   : Single;
  barva     : LongWord;
end;
const
Trojuhelniky: array [0..35] of VLASTNIVERTEX = (
  //predni strana
  (x:  -1.0; y:   1.0; z: -1.0; barva: $001000),
  (x:   1.0; y:  -1.0; z: -1.0; barva: $001000),
  (x:  -1.0; y:  -1.0; z: -1.0; barva: $001000),
  (x:  -1.0; y:   1.0; z: -1.0; barva: $002000),
  (x:   1.0; y:   1.0; z: -1.0; barva: $002000),
  (x:   1.0; y:  -1.0; z: -1.0; barva: $002000),
  . . .
```

```
    . . .
    //dolni strana
    (x:  -1.0; y:  -1.0; z: -1.0; barva: $00B000),
    (x:   1.0; y:  -1.0; z: -1.0; barva: $00B000),
    (x:  -1.0; y:  -1.0; z:  1.0; barva: $00B000),
    (x:   1.0; y:  -1.0; z: -1.0; barva: $00C000),
    (x:   1.0; y:  -1.0; z:  1.0; barva: $00C000),
    (x:  -1.0; y:  -1.0; z:  1.0; barva: $00C000));
D3DFVF_CUSTOMVERTEX = (D3DFVF_XYZ or D3DFVF_DIFFUSE);
```

Program *Rotace krychle* je odvozen od předchozí aplikace vykreslující trojúhelník ve 3D. V programu jsou kromě zadání vrcholů uvedeném ve výpisu 5.7 provedeny následující změny. V metodě *CreateVertexBuffer()* je změněna hodnota násobitele v prvním argumentu ze 3 na 36. V proceduře *VykresleníD3D* je ve volání metody *DrawPrimitive()* změněn typ primitiva z D3DPT_TRIANGLESTRIP na D3DPT_TRIANGLELIST a počet primitiv na 12. Pro vytvoření dojmu, že se jedná opravdu o trojrozměrný objekt, je v programu nastavena pozice kamery tak, aby bylo na objekt nahlíženo lehce z vrchu. Poslední změnou je vytvoření jednoduché animace. Do aplikace je přidána metoda pro rotaci *D3DXMatrixRotationY()*, která následně volána v časovači. Tyto změny jsou vidět v aplikaci *Rotace krychle*, která je na přiloženém CD.

5.5 Kombinování transformací

Program rotující krychle dokáže provádět pouze rotaci kolem osy Y. Chceme-li ale provádět rotace ve směru více os, či dokonce objekt posouvat nebo měnit jeho velikost, nelze matice pro transformace používat tak jednoduše, jako v předchozím případě. K provádění více transformací současně je třeba použít násobení matic. K násobení matic se používá pomocná funkce *D3DXMatrixMultiply()*, která je v SDK DirectX deklarována následovně:

```
D3DXMATRIX * D3DXMatrixMultiply(
  D3DXMATRIX * pOut,
  CONST D3DXMATRIX * pM1,
  CONST D3DXMATRIX * pM2
);
```

Význam argumentů je následující:

- *pOut* – Ukazatel na matici, do které budou uloženy výsledky operace.
- *pM1* – Ukazatel na první matici, která má být při operaci použita.
- *pM2* – Ukazatel na druhou matici, má být při operaci použita.

Dále je k provádění více transformací současně třeba deklarovat funkci, která inicializuje matice na maticovou identitu. Funkce se nazývá *D3DXMatrixIdentity()* a v Diret3D je deklarována následovně:

```
D3DXMATRIX * D3DXMatrixIdentity(
  D3DXMATRIX * pOut
);
```

Jediným argumentem funkce je ukazatel na matici, do které budou vloženy výsledky operace.

Nastavení více transformací objektu současně lze provést způsobem, který je uveden ve výpisu 5.8. Jsou deklarovány matice objektu a dočasné matice. Následně je matice pro objekt inicializována na maticovou identitu. První rotace kolem osy X se provede zavoláním funkce *D3DXMatrixRotationX()* a uložením výsledků této funkce do dočasné matice. Dočasná matice nyní obsahuje hodnoty potřebné pro otočení kolem osy X. Poté je potřeba dostat tento výsledek do matice pro objekt. To provede funkce *D3DXMatrixMultiply()*. Postup naplnění dočasné matice hodnotami pro otočení kolem os Y a Z a předání těchto výsledků do matice pro objekt se opakuje, až do vykonání všech požadovaných transformací. Kompletní kód aplikace demonstrující použití více transformací současně lze nalézt na přiloženém CD.

Výpis 5.8 Použití více transformací současně

```
objekt, docasnaMatice: TD3DMatrix;
. . .
D3DXMatrixIdentity(objekt);
D3DXMatrixRotationX(docasnaMatice, g_pRotaceX);
D3DXMatrixMultiply(objekt, objekt, docasnaMatice);
D3DXMatrixRotationY(docasnaMatice, g_pRotaceY);
D3DXMatrixMultiply(objekt, objekt, docasnaMatice);
D3DXMatrixRotationZ(docasnaMatice, g_pRotaceZ);
D3DXMatrixMultiply(objekt, objekt, docasnaMatice);
g_pZarizeniDirect3D.SetTransform(D3DTS_WORLD, objekt);
```

Důležitou věcí při provádění několika transformací najednou, je jejich pořadí. Provedeme-li rotaci objektu následovanou posunem, bude výsledný efekt úplně jiný, než když provedeme nejdříve posun a poté rotaci. To je způsobeno tím, že při provádění rotace dochází k otáčení celého virtuálního světa, nikoliv pouze objektu. Aplikace, která ukazuje závislost na pořadí transformací je na přiloženém CD.

6 Nasvícení objektů ve scéně

6.1 Zdrojová světla

Ve skutečném světě existují světla v mnoha podobách a lze je vytvářet různými typy zdrojů. Stejně je tomu i v Direct3D. Například klasická žárovka nabízí jiný typ světla, než slunce či reflektor automobilu. Direkt3D obsahuje metody pro reálnou simulaci těchto různých typů světel. Direct3D pracuje se čtyřmi typy zdrojových světel:

- *Bodové světlo* – Má barvu a pozici, nikoliv však směr. Světlo je vyzařováno rovnoměrně všemi směry.
- *Směrové světlo* – Má barvu a směr, nikoliv však zřejmou pozici. Jedná se o paralelní světlo, které vstupuje do scény ze vzdáleného světelného zdroje.
- *Kuželové světlo* – Má barvu, směr i pozici a vyzařuje kužel světla, který je ve středu silnější, směrem k okrajům jeho intenzita slábne.
- *Okolní světlo* – Nachází se všude ve scéně a nemá žádný zdroj. Může přicházet z libovolného množství zdrojů a směrů.

6.2 Odrazná světla

Způsob chování světla v trojrozměrné scéně záleží na několika faktorech. Především na definici objektů ve scéně. Světlo se může například odrážet od lesklých povrchů a způsobovat tím světlá místa nebo se může odrážet od plochých povrchů, kde se pak barva světla a plochy objektu mísí a vzniká odražené světlo. Direct3D pracuje s těmito typy odrazných světel:

- *Rozptýlené* – Výsledné světlo se skládá z barvy materiálu objektu a barvy světla, které na objekt přichází.
- *Zrcadlové* – Výsledné světlo se skládá ze světelného odrazu od lesklých povrchů.
- *Okolní* – Světlo ovlivňuje barvu objektu ve spojení s ostatními typy světel, která jsou odražená nebo vyzařovaná z povrchu objektů.
- *Vyzařující* – Světlo vychází přímo z objektů ve scéně. Nemá odrazné účinky, lze ho tedy klasifikovat jako světlo zdrojové.

6.3 Parametry světel

Konečné nasvícení scény může ovlivnit ještě několik dalších atributů. Jde o následující parametry světel:

- *Umístění* – Jedná se o umístění zdrojového světla ve scéně daného souřadnicemi X, Y a Z. Určuje se pomocí struktury TD3DVECTOR.
- *Směr* – Jedná se o směr, kterým zdrojové světlo prochází scénou. I tento faktor se určuje pomocí struktury TD3DVECTOR.
- *Dosah* – Jedná se o vzdálenost, po kterou světlo dosahuje. Všechny objekty za touto hranicí nejsou nasvíceny.
- *Útlum* – Atribut udává množství světla, které ubývá cestou od světelného zdroje po maximální dosah.

6.4 Definice světla

Vytvoření světla v Diect3D vyžaduje zadání několika informací. Tyto informace jsou uspořádány ve struktuře D3DLIGHT9, která je v Direct3D definována následovně:

```
typedef struct D3DLIGHT9 {
  D3DLIGHTTYPE Type;
  D3DCOLORVALUE Diffuse;
  D3DCOLORVALUE Specular;
  D3DCOLORVALUE Ambient;
  D3DVECTOR Position;
  D3DVECTOR Direction;
  float Range;
  float Falloff;
  float Attenuation0;
  float Attenuation1;
  float Attenuation2;
  float Theta;
  float Phi;
} D3DLIGHT9, *LPD3DLIGHT;
```

Při definování světla musíme nejprve zadat typ světla. To se provede pomocí konstant (D3DLIGHT_POINT, D3DLIGHT_SPOT a D3DLIGHT_DIRECTIONAL). Poté se určí množství a barva reflexního světla (okolní, rozptýlené, zrcadlové) produkované světelným zdrojem. Pokud to daný typ světla vyžaduje, definuje se pozice a směr světelného zdroje. Dále lze definovat útlum a dosah světla.

Je-li světlo nadefinováno musí se do Direct3D přidat pomocí metody objektu zařízení *SetLight()*. Posledním krokem při tvoření osvětlené virtuální scény je zapnutí světla pomocí metody objektu zařízení *LightEnable()*.

6.5 Definice materiálu objektů

Definování zdroje světla je pouze začátek vytvoření reálné osvětlené scény. Objektům ve scéně se musí přiřadit materiál. Materiál určuje, kolik světla a jaká jeho barva na objekt dopadá. Podobně jako u definování světla, existuje i pro materiál struktura v Direct3D nazvaná D3DMATERIAL9. Tuto strukturu definuje SDK DirectX následovně:

```
typedef struct D3DMATERIAL9 {
  D3DCOLORVALUE Diffuse;
  D3DCOLORVALUE Ambient;
  D3DCOLORVALUE Specular;
  D3DCOLORVALUE Emissive;
  float Power;
} D3DMATERIAL9, *LPD3DMATERIAL9;
```

První čtyři proměnné označují typ světla, které materiál odráží. Aby materiál odrážel určitý typ světla, musí mít proměnná pro dané světlo nenulovou hodnotu. Proměnná *Power* určuje intenzitu zrcadlového světla.

Po inicializaci materiálu jej předáme Direct3D pomocí metody objektu zařízení *SetMaterial()*.

6.6 Definice normál

Pro správné nasvícení scény, musí Direct3D vědět, kterým směrem jsou trojúhelníky ve scéně natočeny. Tuto informaci poskytuje právě definice normál. Normála je normalizovaný vektor o velikosti 1. Více o výpočtu a normalizaci normál lze nalézt v [16]. Ukázkové aplikace obsahují informaci o normálách ve vrcholech.

6.7 Použití světel v praxi

Ve výpisu 6.1 je ukázka definice vrcholů, které obsahují informace o normálách. V první řadě je třeba deklarovat nový datový typ vrcholu a definovat vrcholy objektu. Dále je třeba deklarovat konstantu D3DFVF_CUSTOMVERTEX, která obsahuje příznaky D3DFVF_XYZ a D3DFVF_NORMAL určující typ vrcholu. Ve výpisu je ukázána definice přední strany krychle. Kompletní definice vrcholů krychle lze shlédnout na přiloženém CD.

Výpis 6.1 Definice vrcholů krychle

```
type
VLASTNIVERTEX = record
  x, y, z  : Single;
  normala  : TD3DVector;
end;
const
Krychle: array [0..35] of VLASTNIVERTEX = (
  //predni strana
  (x: -1.0; y: 1.0; z: -1.0; normala:(x: 0.0; y: 0.0; z: -1.0)),
  (x: 1.0; y: -1.0; z: -1.0; normala:(x: 0.0; y: 0.0; z: -1.0)),
  (x: -1.0; y: -1.0; z: -1.0; normala:(x: 0.0; y: 0.0; z:-1.0)),
  (x: -1.0; y: 1.0; z: -1.0; normala:(x: 0.0; y: 0.0; z: -1.0)),
  (x: 1.0; y: 1.0; z: -1.0; normala:(x: 0.0; y: 0.0; z: -1.0)),
  (x: 1.0; y: -1.0; z: -1.0; normala:(x: 0.0; y: 0.0; z: -1.0)),
  . . .
D3DFVF_CUSTOMVERTEX = (D3DFVF_XYZ or D3DFVF_NORMAL);
```

6.7.1 Používání vyzařujícího světla

Přiložené CD obsahuje aplikaci ukazující vyzařující světlo na rotující krychli. Objektu chybí jakékoliv detaily. Vyzařující světlo se používá nejsnadněji, jelikož vyžaduje pouze definování materiálu. V programu je vytvořena procedura *InicializaceSvetel*, ve které definujeme světlo a materiál. V tomto případě pouze materiál. Zápis procedury definující materiál je uveden ve výpisu 6.2.

Výpis 6.2 Definice materiálu vyzařujícího světla

```
procedure TForm1.InicializaceSvetel;
var
  materiál : TD3DMaterial9;
begin
  ZeroMemory(@material, SizeOf(material));
  material.Emissive.r := 0.5;
  material.Emissive.g := 0.75;
  material.Emissive.b := 0.5;
  material.Emissive.a := 0.0;
  g_pZarizeniDirect3D.SetMaterial(material);
end;
```

6.7.2 Používání okolního světla

Dalším použitelným typem světla v Direct3D je okolní světlo. Stejně jako u vyzařujícího světla není nutné definovat objekty světel. Použití okolního světla je ukázáno ve výpisu 6.3. Výsledný efekt použití tohoto světla je stejný jako v předchozím případě, pouze zdrojem světla je nyní světlo okolní. Voláním metody objektu zařízení *SetRenderState()* s příznakem D3DRS_AMBIENT a barvou světla, se zapneme okolní světlo. Jak již ale bylo uvedeno, není potřeba definovat žádný objekt světla.

Výpis 6.3 Definice okolního světla

```
procedure TForm1.InicializaceSvetel;
var
  materiál : TD3DMaterial9;
begin
  ZeroMemory(@material, SizeOf(material));
  material.Ambient.r := 0.5;
  material.Ambient.g := 0.75;
  material.Ambient.b := 0.5;
  material.Ambient.a := 0.0;
  g_pZarizeniDirect3D.SetMaterial(material);
  g_pZarizeniDirect3D.SetRenderState(D3DRS_AMBIENT,
    D3DCOLOR_ARGB(0,255, 255, 255));
end;
```

6.7.3 Používání bodového světla

Prvním světlem vyžadujícím objekt světla je světlo bodové. Zde se již musí definovat pozice světla. Příkladem takového světla ve skutečném světě může být žárovka. Definice takové žárovky vypadá následovně (viz. výpis 6.4).

Výpis 6.4 Definice bodového světla

```
procedure TForm1.InicializaceSvetel;
var
  svetlo   : TD3DLight9;
  materiál : TD3DMaterial9;
begin
  ZeroMemory(@svetlo, SizeOf(svetlo));
  svetlo._Type := D3DLIGHT_POINT;
  svetlo.Diffuse.r := 1.0;
  svetlo.Diffuse.g := 1.0;
  svetlo.Diffuse.b := 1.0;
  svetlo.Position.x :=  0.0;
  svetlo.Position.y :=  5.0;
  svetlo.Position.z := -10.0;
  svetlo.Range := 100.0;
  svetlo.Attenuation0 := 1.0;
  g_pZarizeniDirect3D.SetLight(0, svetlo);
  g_pZarizeniDirect3D.LightEnable(0, TRUE);
  ZeroMemory(@material, SizeOf(material));
  material.Diffuse.r := 1.0;
  material.Diffuse.g := 0.3;
  material.Diffuse.b := 1.0;
  material.Diffuse.a := 0.0;
  g_pZarizeniDirect3D.SetMaterial(material);
end;
```

Prvním krokem při definici světla, je jeho typ. V případě bodového světla je tento typ určen konstantou D3DLIGHT_POINT. Jelikož se pracuje s rozptýleným světlem, v proceduře se nastavují barvy pouze po členskou proměnnou *Diffuze*. Nastavením těchto hodnot na 1.0 bude objekt bodového světla vyzařovat čisté bílé rozptýlené světlo. Dalším krokem je umístění světla v prostoru, vzhledem k počátku souřadného systému. Poté je v proceduře definován dosah světla na hodnotu 100. Dále je nastaven útlum

světla (*Attenuation*). Nastavením argumentu *Attenuation0* na hodnotu 1.0 a současně argumentů *Attenuation1* a *Attenuation2* na 0.0 je dosaženo nulového útlumu. Posledním krokem je přiřazení a aktivování světla pomocí metod *SetLight()* a *LightEnable()*.

Po definování objektu světla je třeba vytvořit materiál, který by toto světlo odrážel. Materiál definovaný v proceduře odráží 100 procent červeného a modrého rozptýleného světla ve scéně a 30 procent zelené barvy. Výsledkem je fialová krychle.

6.7.4 Používání směrového světla

Použití směrového světla je podobné použití bodového světla, pouze s rozdílem, že směrové světlo nemá žádnou pozici, pouze směr.

Výpis 6.5 Definice směrového světla

```
procedure TForm1.InicializaceSvetel;
var
  materiál : TD3DMaterial9;
  svetlo   : D3DLIGHT9;
begin
  ZeroMemory(@svetlo, SizeOf(svetlo));
  svetlo._Type  := D3DLIGHT_DIRECTIONAL;
  svetlo.Diffuse.r   := 1.0;
  svetlo.Diffuse.g   := 1.0;
  svetlo.Diffuse.b   := 1.0;
  svetlo.Ambient.r   := 1.0;
  svetlo.Ambient.g   := 1.0;
  svetlo.Ambient.b   := 1.0;
  svetlo.Direction.x  := 0.0;
  svetlo.Direction.y  := 0.0;
  svetlo.Direction.z  := 1.0;
  g_pZarizeniDirect3D.SetLight(0, svetlo);
  g_pZarizeniDirect3D.LightEnable(0, TRUE);
  ZeroMemory(@material, SizeOf(material));
  material.Diffuse.r  := 0.0;
  material.Diffuse.g  := 1.0;
  material.Diffuse.b  := 0.0;
  material.Diffuse.a  := 0.0;
  g_pZarizeniDirect3D.SetMaterial(material);
end;
```

Typ směrového světla je určen konstantou D3DLIGHT_DIRECTIONAL. Světlo přichází do scény ze předu a směřuje dozadu ve směru osy Z. Směr lze měnit zadáním souřadnic ve směru více os.

6.7.5 Používání kuželového světla

Realisticky nejlépe vypadající světlo je světlo kuželové. Z toho i plyne, že jeho použití je také nejobtížnější. Pro programátora vyžaduje množství nastavení. Kromě nastavení barvy, pozice a směru, je třeba zadat světlu také proměnné *Phi*, *Theta*

a *Falloff*. Tyto hodnoty se používají k definici kuželu světla. *Phi* označuje úhel v radiánech, jenž určuje plnou šířku světla, zatímco *Theta* (také v radiánech) označuje šířku menšího vnitřního kuželu. Menší kužel světla způsobuje, že je střed velkého kuželu světlejší než jeho okraje. Proměnná *Falloff* označuje útlum mezi těmito kužely. Jak může být takové světlo definované v praxi, je ukázáno ve výpisu 6.6.

Výpis 6.6 Definice kuželového světla

```
procedure TForm1.InicializaceSvetel;
var
  materiál : TD3DMaterial9;
  svetlo   : D3DLIGHT9;
begin
  ZeroMemory(@svetlo, SizeOf(svetlo));
  svetlo._Type  := D3DLIGHT_SPOT;
  svetlo.Diffuse.r   := 1.0;
  svetlo.Diffuse.g   := 1.0;
  svetlo.Diffuse.b   := 1.0;
  svetlo.Ambient.r   := 1.0;
  svetlo.Ambient.g   := 1.0;
  svetlo.Ambient.b   := 1.0;
  svetlo.Specular.r  := 1.0;
  svetlo.Specular.g  := 1.0;
  svetlo.Specular.b  := 1.0;
  svetlo.Position.x  := 2.0;
  svetlo.Position.y  := 2.0;
  svetlo.Position.z  := 0.0;
  svetlo.Direction.x := -1.0;
  svetlo.Direction.y := -1.0;
  svetlo.Direction.z := 0.0;
  svetlo.Attenuation0:=1.0;
  svetlo.Range       := 50.0;
  svetlo.Phi         := 2.0;
  svetlo.Theta       := 1.0;
  svetlo.Falloff     := 1.0;
  g_pZarizeniDirect3D.SetLight(0, svetlo);
  g_pZarizeniDirect3D.LightEnable(0, TRUE);
  ZeroMemory(@material, SizeOf(material));
  material.Diffuse.r := 0.0;
  material.Diffuse.g := 1.0;
  material.Diffuse.b := 0.0;
  material.Diffuse.a := 0.0;
  material.Ambient.r := 0.0;
  material.Ambient.g := 0.4;
  material.Ambient.b := 0.0;
  material.Ambient.a := 0.0;
  g_pZarizeniDirect3D.SetMaterial(material);
end;
```

V případě kuželového světla definujeme typ světla konstantou D3DLIGHT_SPOT. Definice ostatních proměnných je podobná nastavení předchozích zdrojů světla, pouze přibyla definice vnitřního malého, vnějšího velkého kuželu a útlumu mezi nimi.

Kompletní zdrojové kódy k aplikacím demonstrující osvětlení v mnoha podobách jsou umístěny na přiloženém CD.

7 Textury

7.1 Mapování textur

Objekty vytvořené z polygonů postrádají jakékoliv detaily, které by vyvolaly dojem skutečného objektu. Proto se při programování virtuálních scén na polygony mapují textury. To spočívá v nanášení obrázků na povrch polygonů.

Mapování textur představuje komplikovaný proces, který vyžaduje složité výpočty. Direct3D se však o matematickou stránku věci umí postarat. Programátorovi poté stačí znát pouze několik parametrů a musí Direct3D říct, kde nalezne texturu, kterou chce na polygon nanést.

7.2 Vytvoření povrchu textury

Před nanesením textury na polygon, je třeba texturu nejprve načíst do paměti. Direct3D k tomuto nabízí pomocnou funkci, díky které je načítání bitmapy od paměti velice snadné. Direct3D uchovává textury v samostatném povrchu, který je reprezentován objektem s rozhraním *IDirect3DTexture9*. Deklarace textury potom vypadá následovně:

```
g_pTextura          : IDirect3DTexture9;
```

Po deklarování proměnné, která bude obsahovat texturu, lze k načtení obrázku do povrchu použít funkci *D3DXCreateTextureFromFile()*, kterou Ditect3D definuje následovně:

```
HRESULT D3DXCreateTextureFromFile(
  LPDIRECT3DDEVICE9 pDevice,
  LPCTSTR pSrcFile,
  LPDIRECT3DTEXTURE9 * ppTexture
);
```

Význam argumentů této funkce je následující:

- *pDevice* – Ukazatel na zařízení Direct3D.
- *pSrcFile* – Název a cesta k souboru obsahující obrázek textury.
- *ppTexture* – Adresa, kam má funkce uložit ukazatel na nově vytvořený objekt povrchu textury.

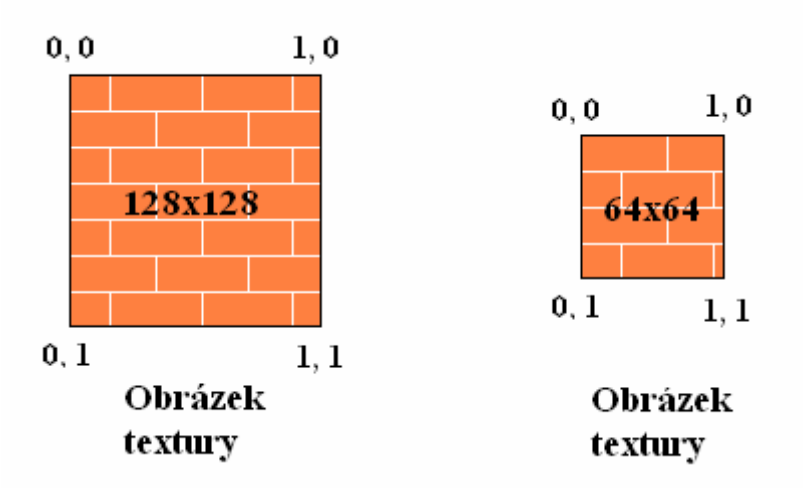
V následujícím výpisu 7.1 je ukázáno volání této metody v programu *Textura*:

Výpis 7.1 Načtení textury ze souboru

```
D3DXCreateTextureFromFile(g_pZarizeniDirect3D,'cihly.jpg',g_pTextura);
```

7.3 Souřadnice textury

Jelikož mohou mít textury různé velikosti, a navíc ve většině případů jsou aplikovány na povrchy, které těmto velikostem neodpovídají, musí Direct3D vědět, jak mapovat obrázek na grafická primitiva. Právě k tomuto účelu se používají souřadnice textury. Bez ohledu na velikost textury se používají souřadnice s hodnotami mezi 0.0 a 1.0, jak je vidět na obrázku 7.1.



Obrázek 7.1 Mapování souřadnic textur

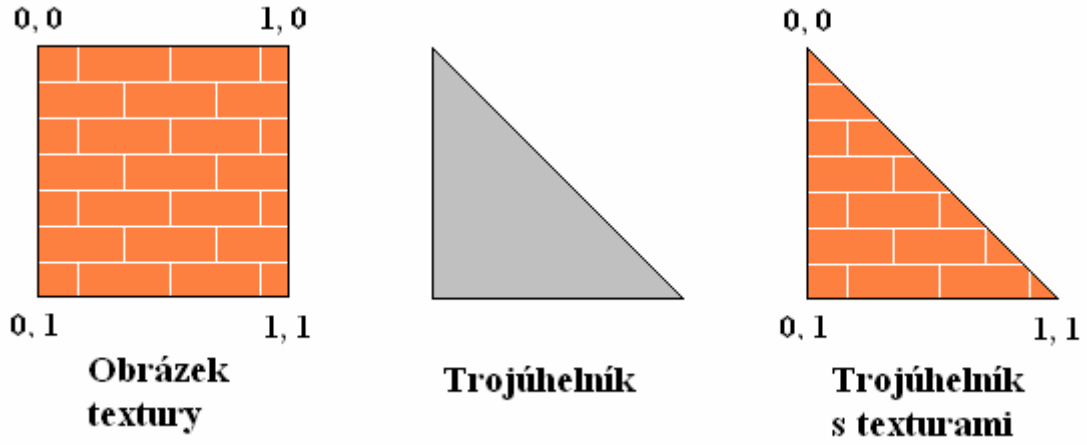
Souřadnice textury umožňují nezabývat se rozměry textury. Direct3D provede mapování za nás, musíme mu ale poskytnout souřadnice textury při definování vrcholů. Souřadnice říkají Direct3D, které části textury mají být aplikovány na konečný tvar. Následující výpis 7.2 deklaruje nový typ vrcholu obsahující navíc souřadnice textury.

Výpis 7.2 Definice vrcholů trojúhelníku

```
type
VLASTNIVERTEX = record
   x, y, z   : Single;
   normala   : TD3DVector;
   tu, tv    : Single;
end;
const
Trojuhelnik: array [0..2] of VLASTNIVERTEX = (
   (x:-1.0;y:1.0;z:-1.0; normala:(x:0.0;y:0.0;z:-1.0);tu:0.0;tv:0.0),
   (x:1.0;y:-1.0;z:-1.0; normala:(x:0.0;y:0.0;z:-1.0);tu:1.0;tv:1.0),
   (x:-1.0;y:-1.0;z:-1.0;normala:(x:0.0;y:0.0;z:-1.0);tu:0.0;tv:1.0));
D3DFVF_CUSTOMVERTEX = (D3DFVF_XYZ or D3DFVF_NORMAL or D3DFVF_TEX1);
```

Vrcholy trojúhelníku definují souřadnice vrcholu, normálu vrcholu a nakonec souřadnice textury. Typ vrcholu určují příznaky D3DFVF_XYZ, D3DFVF_NORMAL A D3DFVF_TEX1, které říkají Direct3D, že je vrchol definován právě souřadnicemi

vrcholu, normálou a souřadnicemi textury. Na obrázku 7.2 je vidět, jak se taková textura aplikuje na daný trojúhelník.

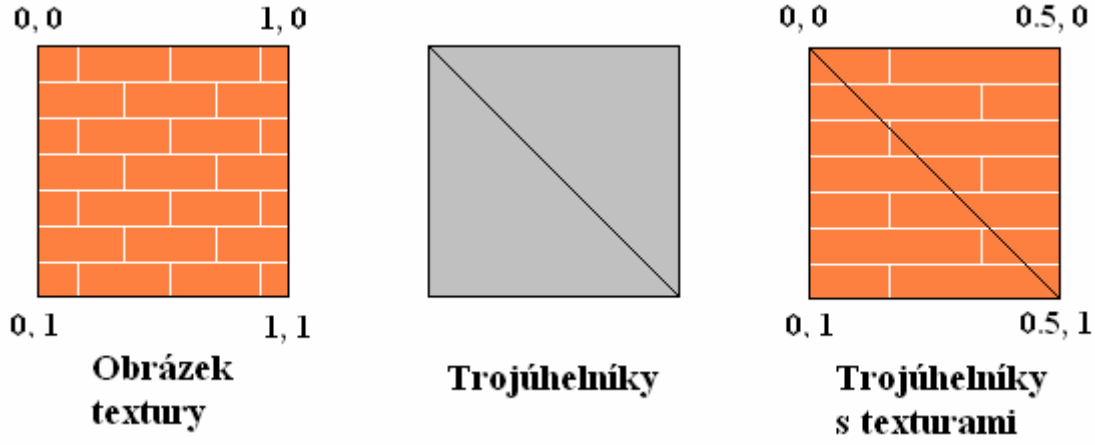


Obrázek 7.2 Aplikování textury na trojúhelník

Podobným způsobem lze mapovat texturu i na čtverec složený z trojúhelníku, popřípadě mapovat pouze polovinu textury na výsledný tvar. Definice takového tvaru se souřadnicemi textury je uvedená ve výpisu 7.3. Postup mapování takového čtverce je ukázán na obrázku 7.3.

Výpis 7.3 Definice vrcholů čtverce

```
const
Ctverec: array [0..5] of VLASTNIVERTEX = (
   (x:-1.0;y:1.0;z:-1.0;normala:(x:0.0;y:0.0;z:-1.0);tu:0.0;tv:0.0),
   (x:1.0;y:-1.0;z:-1.0;normala:(x:0.0;y:0.0;z:-1.0);tu:0.5;tv:1.0),
   (x:-1.0;y:-1.0;z:-1.0;normala:(x:0.0;y:0.0;z:-1.0);tu:0.0;tv:1.0),
   (x:-1.0;y:1.0;z:-1.0;normala:(x:0.0;y:0.0;z:-1.0);tu:0.0;tv:0.0),
   (x:1.0;y:1.0;z:-1.0;normala:(x:0.0;y:0.0;z:-1.0);tu:0.5;tv:0.0),
   (x:1.0;y:-1.0;z:-1.0;normala:(x:0.0;y:0.0;z:-1.0);tu:0.5;tv:1.0));
```



Obrázek 7.3 Čtverec potažený levou polovinou textury

7.4 Vykreslení textury

Po načtení textur do paměti a definování souřadnic textury vrcholu, je třeba nastavit stavy textury. Více se lze o těchto stavech dočíst v dokumentaci k Direct3D [7]. Pro většinu prací s texturami by mělo stačit nastavení obsažené ve výpisu 7.4.

Výpis 7.4 Nastavení stavů textury

```
with g_pZarizeniDirect3D do begin
  SetTextureStageState(0, D3DTSS_COLOROP, D3DTOP_MODULATE);
  SetTextureStageState(0, D3DTSS_COLORARG1, D3DTA_TEXTURE);
  SetTextureStageState(0, D3DTSS_COLORARG2, D3DTA_DIFFUSE);
  SetTextureStageState(0, D3DTSS_ALPHAOP, D3DTOP_DISABLE);
end;
```

Výpis obsahuje čtyři volání metody objektu zařízení *SetTextureStageState()*. Tuto metodu definuje SDK DirectX následovně:

```
HRESULT SetTextureStageState(
  DWORD Stage,
  D3DTEXTURESTAGESTATETYPE Type,
  DWORD Value
);
```

Její argumenty mají tento význam:

- *Stage* – ID úrovně textury, může nabývat hodny od 0 do 7.
- *Type* – Stav textury, který má být nastaven.
- *Value* – Hodnota, která má být danému stavu přiřazena.

7.5 Nastavení textury

Posledním krokem před vykreslením je předání textury systému Direct3D. K tomu Direct3D nabízí metodu objektu zařízení *SetTexture()*, která je v Direct3D definována následovně:

```
HRESULT SetTexture(
  DWORD Stage,
  LPDIRECT3DBASETEXTURE9 pTexture
);
```

Argumenty metody mají následující význam:

- *Stage* – Úroveň textury, pro kterou chceme texturu nastavit.
- *pTexture* – Ukazatel na objekt rozhraní IDirect3DTexture asociovaný s obrázkem textury.

Samotné volání metody *SetTexture()* pak v aplikaci vypadá následovně:

Výpis 7.5 Předání textury systému Direct3D

```
SetTexture(0, g_pTextura);
```

7.6 Aplikace Textura

Program mapující texturu na polygony složené do krychle je na přiloženém CD. Aplikace nejprve texturu načte ze souboru, který je umístěn v kořenovém adresáři aplikace. Šířka a výška textury musí dohromady dávat druhou mocninu a zpravidla by neměla být větší než 256x256 pixelů. Dále je textura mapována na krychli, která rotuje kolem osy Y v trojrozměrném prostoru. Aplikace se ukončí stisknutím klávesy ESC.

7.7 Průhlednost textury

Důležitou technikou při programování grafických aplikací je implementace průhlednosti. Průhlednost umožňuje aplikacím, aby zobrazovaly bitmapy, které nejsou obdélníkové nebo obsahují oblasti, skrz které je vidět.

Nejprve je třeba takovou bitmapu vytvořit. V použité textuře se pomocí konkrétní barvy označí oblasti, kde má být bitmapa průhledná. Poté se Direct3D řekne, která barva je zvolena pro označení průhlednosti a jak bitmapu vykreslit. Nakonec Direct3D bitmapu vykreslí na obrazovku.

Direct3D implementuje průhlednost prostřednictvím hodnot alfa. Hodnota 0 říká, že barva je zcela průhledná a hodnota 255 naopak, že je barva neprůhledná.

Postup načítání textury s implementací průhlednosti je stejný, jako při načítání jednoduchých textur, s tím rozdílem, že se k načtení volá funkce *D3DXCreateTexture-FromFileEx()*, která je v Direct3D definována následovně:

```
HRESULT D3DXCreateTextureFromFileEx(
  LPDIRECT3DDEVICE9 pDevice,
  LPCTSTR pSrcFile,
  UINT Width,
  UINT Height,
  UINT MipLevels,
  DWORD Usage,
  D3DFORMAT Format,
  D3DPOOL Pool,
  DWORD Filter,
  DWORD MipFilter,
  D3DCOLOR ColorKey,
  D3DXIMAGE_INFO * pSrcInfo,
  PALETTEENTRY * pPalette,
  LPDIRECT3DTEXTURE9 * ppTexture
);
```

Význam argumentů, které jsou využity pro ukázkovou aplikaci je následující:

- *pDevice* – Ukazatel na zařízení Direct3D, které je spojeno s danou texturou.
- *pSrcFile* – Cesta k textuře.

- *Width* – Šířka textury. Lze použít konstantu D3DX_DEFAULT pro použití šířky definované v souboru.
- *Height* – Výška textury. Lze použít konstantu D3DX_DEFAULT pro použití šířky definované v souboru.
- *Format* – Pixelový formát.
- *Pool* – Třídy paměti textury. Pro účel aplikace je použito D3DPOOL_MANAGED.
- *Filter* – Typ použitého filtrování.
- *MipFilter* – Další nastavení filtru.
- *ColorKey* – 32 bitová barevná hodnota označující průhlednou barvu. Tato barva je ve formátu ARGB.
- *ppTexture* – Adresa ukazatele na objekt textury.

Příklad volání funkce *D3DXCreateTextureFromFileEx()* je ukázán ve výpisu 7.6.

Výpis 7.6 Načtení textury ze souboru

```
D3DXCreateTextureFromFileEx(
  g_pZarizeniDirect3D, 'stonesr.bmp',
  D3DX_DEFAULT, D3DX_DEFAULT, D3DX_DEFAULT, 0,
  D3DFMT_A8R8G8B8, D3DPOOL_MANAGED, D3DX_FILTER_TRIANGLE,
  D3DX_FILTER_TRIANGLE, D3DCOLOR_RGBA(255,0,0,255),
  nil, nil, g_pTextura);
```

Dalším krokem je nastavení stavů textury, aby podporoval míchání alfa. Více o stavech se lze opět dočíst v dokumentaci k Direct3D [7]. Ve většině případů postačí následující nastavení stavů textury (viz výpis 7.7).

Výpis 7.7 Nastavení stavů textury

```
with g_pZarizeniDirect3D do begin
  SetRenderState(D3DRS_ALPHATESTENABLE, iTRUE);
  SetRenderState(D3DRS_ALPHAREF, iTRUE);
  SetRenderState(D3DRS_ALPHAFUNC, D3DCMP_GREATEREQUAL);
end;
```

Změníme-li v nastavení stavů textur v posledním volání metody *SetRenderStage State()* s parametrem D3DTSS_ALPHAOP druhý parametr z D3DTOP_DISABLE na D3DTOP_SELECTARG1, docílíme zajímavého efektu, který implementace průhlednosti nabízí. Na CD doplňujícím diplomovou lze nalézt i tuto aplikaci, která rozšiřuje aplikaci předchozí. Jako textura je použitá upravená bitmapa z aplikace *Textura*, která má uprostřed červené kolečko. Právě tato červená barva označuje průhledné části textury. Průhledná barva se nastavuje makrem D3DCOLOR_RGBA(255,0,0,255) v metodě pro načtení textury.

8 Míchání alfa a mlha

8.1 Míchání alfa

Další implementací průhlednosti je použití průhlednosti pomocí míchání alfa. Jelikož může nastat situace, kdy je třeba dosáhnout stavu někde mezi viditelnou a neviditelnou oblastí. Například, když chceme při programování virtuální scény, ve které jsou okna s odstínem, aby všechny objekty za těmito okny měli stejný barevný tón, jako mají právě tyto okna.

Podobně jako v předchozím případě (Průhlednost textury) využívá tato technika k určení průhlednosti hodnot alfa. Hodnota 0 řekne Direct3D, že objekt je průhledný, hodnota 255 naopak, že je objekt zcela neprůhledný. Podle uchovávání údajů o hodnotách alfa rozdělujeme na tyto typy míchání alfa:

- Alfa vrcholu.
- Alfa materiálu.
- Alfa textury.
- Alfa snímkového bufferu.
- Alfa cílového vykreslení.

Například v případě míchání alfa pro vrcholy uchovává údaje o hodnotách alfa vrchol, zatímco u míchání alfa pro textury je nositelem její bitmapa. V ukázkovém programu je nositelem hodnot alfa vrchol.

Pro práci s tímto mícháním, je třeba nejdříve vytvořit datový typ vrcholu, který dokáže uchovávat hodnoty alfa. Dalším krokem je definice dat vrcholu čtverce.

Výpis 8.1 Definice vrcholů čtverce

```
type
VLASTNIVERTEX = record
  x, y, z   : Single;
  normala   : TD3DVector;
  barva     : LongWord;
  tu, tv    : Single;
end;
const
plnyCtverec: array [0..5] of VLASTNIVERTEX = (
  (x:  -1.0; y:   1.0; z: 1.0; normala:(x: 0.0; y: 0.0; z: -1.0);
    barva: $FFFFFFFF; tu: 0.0; tv: 0.0),
  (x:   1.0; y:  -1.0; z: 1.0; normala:(x: 0.0; y: 0.0; z: -1.0);
    barva: $FFFFFFFF; tu: 1.0; tv: 1.0),
  (x:  -1.0; y:  -1.0; z: 1.0; normala:(x: 0.0; y:0.0; z:-1.0);
    barva: $FFFFFFFF; tu: 0.0; tv: 1.0),
```

```
    (x:  -1.0; y:   1.0; z: 1.0; normala:(x: 0.0; y:0.0; z:-1.0);
      barva: $FFFFFFFF; tu: 0.0; tv: 0.0),
    (x:   1.0; y:   1.0; z: 1.0; normala:(x: 0.0; y:0.0; z:-1.0);
      barva: $FFFFFFFF; tu: 1.0; tv: 0.0),
    (x:   1.0; y:  -1.0; z: 1.0; normala:(x: 0.0; y:0.0; z:-1.0);
      barva: $FFFFFFFF; tu: 1.0; tv: 1.0));
  D3DFVF_CUSTOMVERTEX = (
    D3DFVF_XYZ or D3DFVF_NORMAL or D3DFVF_DIFFUSE or D3DFVF_TEX1);
```

Data definující barvu mají formát ARGB, což znamená že jednotlivými argumenty této barvy jsou hodnota alfa a množství červené, zelené a modré barevné složky. Všechny hodnoty alfa jsou rovny $FF, což označuje v tomto případě neprůhlednost vrcholu. V případě použití míchání alfa je třeba určit typ vrcholu příznaky, které Direct3D říkají, že se pracuje s netransformovanými souřadnicemi, normálou, mícháním barvy vrcholu a souřadnicemi textury.

Dalším krokem je nastavení stavů vykreslení pro míchání alfa. Zde opět ve většině případů vystačí následující nastavení.

Výpis 8.2 Nastavení stavů vykreslení pro míchání alfa

```
  with g_pZarizeniDirect3D do begin
    SetRenderState(D3DRS_ALPHABLENDENABLE, iTRUE);
    SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
    SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCCOLOR);
  end;
```

První řádek zapne míchání alfa. Následující dva řádky udávají, jak se hodnoty alfa mají použít k míchání zdrojových a cílových barev. Zdrojová barva je barva pixelu, který má být vykreslen, zatímco cílová barva je barva již existujícího pixelu na scéně.

8.2 Aplikace Míchání alfa

Po spuštění programu se na obrazovce zobrazí dva čtverce, z nichž je jeden průhledný a druhý neprůhledný. Průhledný čtverec je reprezentován texturu znázorňující hladinu vody. Neprůhledný čtverec znázorňuje textura kamenné zdi. Oba čtverce se otáčí, díky čemuž je jasně vidět funkce ukázkového programu, tedy demonstrace průhlednosti pomocí míchání alfa. Kompletní zdrojový kód si lze prohlédnout na přiloženém CD.

8.3 Mlha

Další grafickou technikou, kterou Direct3D nabízí, je používání mlhy. Ani ve skutečném světe neexistuje stoprocentní viditelnost, jelikož vzduch kolem nás není

zcela průhledný. To je způsobeno drobnými částečkami, které vzduch obsahuje. Vzdálené objekty se poté zdají být jakoby zastřené.

Programátoři trojrozměrných aplikací používají mlhu nejenom proto, aby výsledná scéna vypadala realističtěji, ale také pro omezení množství objektů, které musí aplikace vykreslit. Jelikož jsou vzdálené objekty zastřené mlhou, nemusí je Direct3D zpracovávat a tím se urychlí proces vykreslování.

Direct3D dokáže efekt mlhy dobře simulovat. Nabízí k tomu několik způsobů, jak mlhu do scény dodat. Zde je ukázán pouze jeden způsob použití mlhy, který ovšem vystačí ve většině případů práce s mlhou. V dokumentaci k Direct3D [7] lze nalézt další typy a způsoby používání mlhy.

8.3.1 Přidání mlhy do scény

Nejprve je třeba zapnout míchání mlhy, k čemuž je třeba zavolat metodu objektu zařízení *SetRenderState()* s argumenty D3DRS_FOGENABLE a iTRUE. Dále je třeba nastavit režim mlhy. Režim mlhy určuje typ výpočtů které má Direct3D použít k jejímu vykreslení. Direct3D používá tři typy mlhy: lineární mlhu a dva typy exponenciálních mlh. Tyto režimy se určují pomocí konstant D3DFOG_LINEAR, D3DFOG_EXP a D3DFOG_EXP2. Pomocí metody *SetRenderState()* se pak daný typ mlhy nastaví. Mlha může mít libovolnou barvu. K nastavení barvy mlhy se opět požívá metoda *SetRenderState()*. Nyní však s argumenty D3DRS_FOGCOLOR a požadovanou barvou. Posledním nastavením je definování hustoty mlhy. To se opět provede zavoláním metody *SetRenderState()* s argumenty D3DRS_FOGDENSITY a množstvím mlhy. Kompletní přidání mlhy do scény vypadá následovně.

Výpis 8.3 Definice mlhy

```
hustota := 0.1;
with g_pZarizeniDirect3D do begin
  SetRenderState(D3DRS_LIGHTING, iTrue);
  SetRenderState(D3DRS_FOGENABLE, iTRUE);
  SetRenderState(D3DRS_FOGVERTEXMODE, D3DFOG_EXP2);
  SetRenderState(D3DRS_FOGCOLOR, D3DCOLOR_ARGB(255,255,250,250));
  SetRenderState(D3DRS_FOGDENSITY, PDWORD (@hustota)^);
end;
```

8.4 Aplikace Mlha

Aplikace *Mlha* vykresluje vzdalující se objekt, který se ztrácí v bílé mlze. Vzdalování objektu a následné přiblížení je prováděno v cyklu. Aplikace se ukončí klávesou ESC. I zdrojový kód tohoto programu lze nalézt na CD k diplomové práci.

9 Síťový model objektu

Nyní se dostáváme k další grafické technice knihovny Direct3D, kterou je načítání a vykreslení složitých objektů. Tyto síťové objekty, někdy označované jako drátěné, jsou obvykle modelovány ve speciálním softwaru, jako je například 3D Studio Max, Blender, Cinema4D či Rhino3D. V tomto speciálním softwaru vytvořený 3D objekt, je uložen do souboru, který lze následně konvertovat do souboru s příponou .x. Tento soubor obsahuje veškeré informace potřebné pro vykreslení těchto drátěných 3D objektů ve virtuálním světě. Objekt je tvořen sítí polygonů, které dokáže Direct3D pomocí svých funkcí ze souboru načíst a dále s nimi pracovat. Postup vykreslení objektu generovaného v 3D softwaru je následující.

- Vytvoření a uložení daného objektu
- Konvertování do x-souboru
- Načtení objektu z x-souboru
- Vykreslení objektu
- Uvolnění alokované paměti

9.1 Vytvoření objektu

K vytvoření 3D objektu existuje mnoho 3D editorů. Všechny mají jednu společnou vlastnost, pracuje se v nich v 3D prostoru, ale liší se hlavně kvalitou, efektivitou práce a samozřejmě cenou. Jedním typem takovéhoto softwaru je 3D Studio Max, který je momentálně asi jeden z nejpoužívanějších a nejprodávanějších 3D nástrojů, k tvorbě her a animací. Více se lze o tomto programu dočíst zde [2].

9.2 Vytvoření x-souboru

Máme-li vytvořený a uložený objekt, je třeba vytvořit pomocí konvertoru soubor s příponou *x*. Tento program (conv3DS) lze stáhnou na internetových stránkách [10].

9.3 Načtení objektu

Před zobrazením objektu na obrazovku je třeba nejprve tento objekt načíst do paměti. Díky Direct3D a jeho pomocným funkcím je načítání objektu z x-souboru

relativně snadné. Direkt3D k tomu účelu používá metodu *D3DXLoadMeshFromX()*, kterou definuje následovně:

```
HRESULT D3DXLoadMeshFromX(
  LPCTSTR pFilename,
  DWORD Options,
  LPDIRECT3DDEVICE9 pD3DDevice,
  LPD3DXBUFFER * ppAdjacency,
  LPD3DXBUFFER * ppMaterials,
  LPD3DXBUFFER * ppEffectInstances,
  DWORD * pNumMaterials,
  LPD3DXMESH * ppMesh
);
```

Význam nejdůležitějších argumentů metody je uveden zde:

- *pFileName* – Název souboru obsahující informace o objektu, souboru s příponou .x.
- *pD3DDevice* – Ukazatel na zařízení, které bude použito pro vykreslení.
- *ppMaterials* – Ukazatel na buffer obsahující materiálová data.
- *pNumMaterials* – Ukazatel na počet položek, které obsahuje pole ppMaterials.
- *ppMesh* – Adresa, kam má funkce uložit ukazatel na nově vytvořený objekt.

Po načtení objektu tvořeného sítí a získání informací o materiálu, je potřeba získat z materiálového bufferu další vlastnosti materiálu a informace pro mapování textur. K tomuto účelu poslouží zavolání metody *GetBufferPointer*, jak je vidět v části kódu ve výpisu 9.1. Další řádky kódu slouží k vytvoření nové sítě objektu a textur, jejichž počet odpovídá konečnému počtu částí sítě (materiálů), ze které je objekt sestaven.

Výpis 9.1 Načtení 3D objektu ze souboru

```
if FAILED(D3DXLoadMeshFromX('tiger.x', D3DXMESH_SYSTEMMEM,
  g_pZarizeniDirect3D, nil, @MaterialBuffer, nil,
  @g_pCisloMaterialu, g_pSit)) then exit;
Materialy := MaterialBuff€er.GetBufferPointer;
GetMem(g_pSitMaterialu, SizeOf(TD3DMaterial9)*g_pCisloMaterialu);
GetMem(g_pSitTextury,SizeOf(IDirect3DTexture9)*g_pCisloMaterialu);
ZeroMemory(g_pSitTextury,SizeOf(IDirect3DTexture9)*g_pCisloMaterialu);
```

Pro každou část sítě je třeba udělat tyto následující kroky. Zkopírovat materiál, nastavit okolní barvu materiálu, a přiřadit danému materiálu texturu. Tyto kroky by v programu vypadali následovně.

Výpis 9.2 Nastavení materiálu sítě

```
while (i < g_pCisloMaterialu) do begin
  g_pSitMaterialu[i] := Materialy[i].MatD3D;
  g_pSitMaterialu[i].Ambient := g_pSitMaterialu[i].Diffuse;
  g_pSitTextury[i] := nil;
  if (Materialy[i].pTextureFilename <> nil) and
    (StrLen(Materialy[i].pTextureFilename) > 0) then
      D3DXCreateTextureFromFile(g_pZarizeniDirect3D,
  Materialy[i].pTextureFilename,
```

```
    g_pSitTextury[i]);
    Inc(i);
  end;
  MaterialBuffer:= nil;
```

9.4 Vykreslení objektu

Proces vykreslení je rozdělen do podmnožin, pro každý materiál, který byl načten jako část sítě. Tyto podmnožiny reprezentují „oka“ sítě, která jsou postupně vykreslována a je na ně nanášena textura.

Výpis 9.3 Vykreslení 3D objektu

```
  while(i < g_pCisloMaterialu) do begin
    g_pZarizeniDirect3D.SetMaterial(g_pSitMaterialu[i]);
    g_pZarizeniDirect3D.SetTexture(0, g_pSitTextury[i]);
    g_pSit.DrawSubset(i);
    Inc(i);
  end;
```

9.5 Uvolnění objektů

Po skončení programu, již jako obvykle, musí program uvolnit paměť. Jak je známo objekty se uvolňují v opačném pořadí, než byly vytvářeny. Toto uvolnění je ukázáno v následujícím výpisu 9.4.

Výpis 9.4 Uvolnění objektů

```
  if assigned(g_pSitMaterialu) then g_pSitMaterialu:= nil;
  if assigned(g_pSitTextury) then
  begin
    i:= 0;
    while(i < g_pCisloMaterialu) do begin
      if assigned(g_pSitTextury[i]) then g_pSitTextury[i]:= nil;
      Inc(i);
    end;
    g_pSitTextury:= nil;
  end;
  if assigned(g_pSit) then g_pSit:= nil;
  if assigned(g_pZarizeniDirect3D) then g_pZarizeniDirect3D:=nil;
  if assigned(g_pDirect3D) then g_pDirect3D:=nil;
```

9.6 Aplikace vykreslení objektu načteného z x-soboru

Tak, jako ke všem předchozím aplikacím, je i k této kompletní zdrojový kód na přiloženém CD. Program vykresluje objekt načtený z x-souboru. Tímto objektem je tygr, jehož x-soubor společně s texturou je dispozici v dokumentaci k Direct3D. Objekt se ve scéně otáčí pro získání lepší vize. Ukončení programu se provede klávesou ESC.

10 Text ve scéně

Tato kapitola se věnuje použití 2D textu v trojrozměrné scéně. To může být užitečné například při zobrazování statistik či menu hry na obrazovku. V následující části je popsáno, jak se u přidání textu do trojrozměrné scény postupuje.

10.1 Vytvoření fontu.

Prvním krokem před vypsáním textu na obrazovku je vytvoření fontu. K tomu slouží v SDK DirectX metoda *D3DXCreateFont()*, kterou definuje následovně:

```
HRESULT D3DXCreateFont(
  PDIRECT3DDEVICE9 pDevice,
  INT Height,
  UINT Width,
  UINT Weight,
  UINT MipLevels,
  BOOL Italic,
  DWORD CharSet,
  DWORD OutputPrecision,
  DWORD Quality,
  DWORD PitchAndFamily,
  LPCTSTR pFacename,
  LPD3DXFONT * ppFont
);
```

Význam důležitých argumentů metody *D3DXCreateFont()* je následující:

- *pDevice* – Ukazatel na zařízení Direct3D.
- *Height* – Výška znaku.
- *Width* – Šířka znaku.
- *MipLevels* – Počet mipmap úrovní. Hodnota 0 generuje úrovně automaticky.
- *Italic* – Hodnota TRUE znamená, že písmo bude psáno kurzívou.
- *ppFont* – Vrací ukazatel na rozhraní *ID3DXFont*, reprezentující vytvořený objekt fontu.

V aplikaci může definování nového fontu vypadat následovně (viz výpis 10.1). Nesmíme zapomenou, že je třeba proměnnou nejdříve deklarovat a inicializovat. Proměnná *g_pFont* je typu *ID3DFont*.

Výpis 10.1 Definice fontu

```
g_pFont: ID3DXFont = nil;
. . .
Result := D3DXCreateFontW(pd3dDevice, 15, 0, FW_BOLD, 1, FALSE,
  DEFAULT_CHARSET, OUT_DEFAULT_PRECIS, DEFAULT_QUALITY,
  DEFAULT_PITCH or FF_DONTCARE, 'arial', g_pFont);
if Failed(Result) then Exit;
```

10.2 Zobrazení textu

Když máme nadefinovaný font, stačí Direct3D říct, jak a na jaké místo na obrazovce chceme daný text zobrazit. To lze udělat dvěmi způsoby. Buď k vykreslení použít tzv. textového pomocníka, který k vykreslení textu používá funkci *DrawTextLine()* nebo definovat přesnou pozici na obrazovce pomocí souřadnic a text vykreslit pomocí funkce *DrawTextW()*.

První způsob definování pozice a vykreslení textu na tuto pozici je pomocí textového pomocníka.

Výpis 10.2 Vykreslení textu pomocí textového pomocníka

```
procedure VykresliText;
var
  txtHelper: CDXUTTextHelper;
begin
  txtHelper := CDXUTTextHelper.Create(g_pFont, g_pTextSprite, 15);
  txtHelper._Begin;
  txtHelper.SetInsertionPos(5, 5);
  txtHelper.SetForegroundColor(D3DXColor(1.0, 1.0, 0.0, 1.0));
  txtHelper.DrawTextLine('Zobrazený text');
  txtHelper.DrawTextLine(DXUTGetFrameStats(TRUE));
  txtHelper.DrawTextLine(DXUTGetDeviceStats);
  txtHelper._End;
  txtHelper.Free;
end;
```

Prvním krokem je deklarace textového pomocníka. Funkce *CDXUTTextHelper.Create()* poté pomocníka vytvoří a zpřístupní tak jeho funkce. Následující řádky nastaví počáteční pozici, barvu textu a vykreslí ho pomocí funkce *DrawTextLine()*.Poslední řádek uvolní alokovanou paměť textového pomocníka.

Direct3D nabízí metody pro vykreslení počtu snímků za sekundu (FPS) *DXUTGetFrameStats(TRUE)* či pro zjištění grafického adaptéru *DXUTGetDeviceStats*.

Druhý zmiňovaný způsob vykreslení textu na obrazovku je ukázán ve výpisu 10.3.

Výpis 10.3 Vykreslení textu

```
procedure VykresliText2;
var
  rc: TRect;
begin
  SetRect(rc, 200, 100, 0, 0);
  g_pFont.DrawTextW(nil, DXUTGetFrameStats(true), -1, @rc,DT_NOCLIP,
    D3DXColorToDWord(D3DXColor(1.0, 1.0, 0.0, 1.0)));
  SetRect(rc, 20, 150, 0, 0);
  g_pFont.DrawTextW(nil, DXUTGetDeviceStats, -1, @rc, DT_NOCLIP,
    D3DXColorToDWord(D3DXColor(1.0, 1.0, 0.0, 1.0)));
  SetRect(rc, 50, 200, 0, 0);
  g_pFont.DrawTextW(nil, 'Zobrazený text', -1, @rc, DT_NOCLIP,
    D3DXColorToDWord(D3DXColor(1.0, 1.0, 0.0, 1.0)));
end;
```

Zde je definována proměnná *rc* typu *TRect*, pomocí které bude možné vykreslovat text na konkrétní pozici. Řádky procedury jsou celkem průhledné. Nejprve se nastaví pozice výpisu textu a poté je pomocí metody *DrawTextW()* objektu fontu vykreslen požadovaný text v dané barvě.

10.3 Aplikace Text

Na CD přiloženém k diplomové práci, lze nalézt program *Text*, který demonstruje použití textu v grafických aplikacích. Aplikace v okně vykresluje počet snímků za sekundu (FPS) a typ grafického adaptéru pomocí obou, zde probraných metod vykreslení textu. Program se ukončí stisknutím klávesy ESC.

11 Porovnání DirectX a OpenGL

Porovnat tyto dvě knihovny vyvinuté pro psaní grafických aplikací a říci o jedné z nich, že je lepší než ta druhá, není jednoduché a ve své podstatě ani možné. Nicméně tato kapitola shrnuje základní rysy, klady a zápory těchto grafických knihoven. Je na každém ze čtenářů diplomové práce, aby rozhodl, zda ta či ona knihovna je pro něj vyhovující. Je zřejmé, že vývojář grafických aplikací pracující pod operačním systémem Linux, zvolí grafickou knihovnu OpenGL. Tím se ovšem již dostáváme k jedné z vlastností OpenGL. V následující části jsou tedy probrány jednotlivé vlastnosti těchto světově uznaných standardů pro vytváření grafických aplikací.

11.1 Knihovna OpenGL

Zkratka OpenGL pochází z anglického *Open Graphics Library*. Knihovna původně vznikla pro potřebu zobrazovat jednoduché grafické prvky na obrazovku. V současné době se využívá pro aplikace typu CAD a pro zobrazení virtuální reality. Rozvoj OpenGL zajišťuje firma Silicon Graphics (SGI) ve spolupráci s firmami, jako jsou například nVidia, ATI, HP, IBM, Apple, které podle potřeby do knihovny přidávají vlastní funkce. Knihovna OpenGL byla navržena tak, aby byla přenositelná a nezávislá na konkrétní platformě. Současná veze rozhraní je OpenGL 2.1.

Jak bylo již v úvodu kapitoly řečeno, první a důležitou vlastností grafické knihovny OpenGL je nezávislost na operačním systému a na hardwaru. Programovat grafické aplikace lze jak pod operačním systém Microsoft Windows, tak pod operačním systémem Linux. Kromě implementací vestavěných na grafické kartě existují také softwarové implementace, které umožňují OpenGL používat i na hardwaru, který ho sám o sobě nepodporuje. To ovšem přináší jisté snížení výkonu. Omezení OpenGL může spočívat v tom, že mu chybí příkazy pro práci s komplexními objekty. OpenGL nepodporuje objektově orientované programování. Dalším z rysů je podpora architektury klient-server a síťového zpracování. Knihovna patří mezi rozhraní nižší úrovně. Představuje vrstvu, která je nezávislá na možnostech grafického podsystému. OpenGL nepodporuje práci s grafickými objekty a soubory.

11.2 Knihovna Direc3D

Výhodou Direct3D oproti OpenGL je přítomnost ostatních DirectX knihoven pro práci se zvukem, vstupními zařízeními či sítí, které dohromady tvoří ucelený balík pojmenovaný SDK DirectX. Nevýhodou zase je, že DirectX je produktem firmy Microsoft Windows a tudíž pracuje pouze na platformě Windows. Další výhodou Direct3D může být, že se do něj přidávají nové funkce, u nichž není podstatné, zda je grafická karta podporuje. V případě, že je nepodporuje, mohou se bez nejmenších požadavků na programátora aplikovat softwarově.

Direct3D podporuje objektově orientované programování. Scéna může být organizována do hierarchických bloků. Dobře napsaný objektově orientovaný program je přehlednější a snadněji modifikovatelný než strukturovaný. Obecně je ovšem kód psaný v Direct3D delší než kód napsaný v OpenGL.

OpenGL i Direct3D jsou uznány za standard grafického rozhraní v oblasti výpočetní techniky. Následující tabulka 11.1 shrnuje základní rysy knihovny OpenGL a Direct3D.

Tabulka 11.1 Základní rysy knihovny OpenGL a Direct3D

OpenGL	***Direct3D***
Nezávislé na platformě a HW	Pouze pod operačním systémem Windows
Strukturované programování	Objektově orientované programování
Pouze pro rending	Součástí uceleného balíku DirectX
Pouze 10 % her programováno v OpenGL	90 % her programováno v Direct3D
Nepodporuje práci s grafickými objekty	Práce s grafickými objekty

Závěr

Diplomová práce se zabývala možnostmi grafické knihovny Direct3D. Podle zadání bylo vytvořeno několik ukázkových aplikací demonstrujících použití této knihovny. K aplikacím byla vytvořena stručná dokumentace, která by měla sloužit programátorům jako příručka. V závěru práce bylo provedeno porovnání grafické knihovny Direct3D s knihovnou OpenGL.

Jelikož jsou Direct3D určené k programování v programovacím prostředí C++, muselo být řešeno použití knihovny v prostředí Borland Delphi. Tento problém byl vyřešen pomocí přeprogramovaných *unitů*, které jsou volně ke stažení na internetu [1]. Avšak to je víceméně vše, co lze na internetu k programování Direct3D v Delphi získat. Proto bych největší přínos mé diplomové práce viděl tom, že zde byla vytvořena dokumentace, která se věnuje základům programování počítačové grafiky v Direct3D s využitím programovacího prostředí Borland Delphi.

Zdokumentována a na ukázkové aplikaci předvedena byla technika vytvoření objektu a objektu zařízení Direct3D. Jelikož každé programování v Direct3D vyžaduje vytvoření těchto objektů, byla tato část programování zařazena na začátek praktické části. Aplikace slouží jako základ ke všem následujícím. První z nich je požívání povrchů v Direct3D. V práci byly popsány jednotlivé povrchy, se kterými Direct3D pracuje a díky kterým jsou animace plynulé a bez nežádoucího blikání. Byla naprogramována aplikace, která přenáší data mezi povrchy. Dále byly vytvořeny aplikace, které vykreslují základní grafická primitiva, která Direct3D používá k vykreslení všech objektů. Důležitá technika pro manipulaci s objekty je transformace objektů. Tomuto tématu se věnuje další kapitola, ke které bylo naprogramováno několik aplikací ukazující transformace objektu. V práci byly popsány typy světel používané v Direct3D a ke každému z nich byl vytvořen ukázkový program. Jedna kapitola byla také věnována mapování textur. Poslední zajímavou technikou, která byla v práci popsána je načtení a zobrazení objektu, vytvořeného v 3D Studiu Max.

Možnosti knihovny Direct3D jsou obrovské. Není v možnostech jedné diplomové práce vytvořit dokumentaci s ukázkovými programy, které by komplexně popisovaly sílu celé této knihovny. Přesto byla vytvořena sada programů, která programátory naučí základní techniky programování grafiky v Direct3D.

Seznam použité literatury

[1] 3D Scéna – Začínáme s 3D Studio Max [online]. datum publikace 15.3.2005, [cit. 2007-4-22] <http://www.3dscena.cz/art/3dscena/zaciname-s-3ds-max.html>.

[2] Autodesk [online]. c2007, [cit. 2007-4-23]. <http://www.autodesk.cz>.

[3] BARKOVOY, A. Clootie graphics pages [online]. [cit. 2007-5-8]. <http://www.clootie.ru/>.

[4] Builder – Informační server o programování [online].[cit. 2006-12-21]. <http://www.builder.cz>.

[5] Centrum Počítačové Grafiky a Vizualizace Dat [online]. poslední aktualizace 5.10. 2006, [cit. 2007-1-12]. <http://herakles.zcu.cz/>

[6] Microsoft [online]. c2007, [cit. 2006-6-9]. <http://www.microsoft.com>.

[7] Microsoft DirectX SDK (June 2006). DirectX Documentation for C++ [počítačový program]. 2006-6-9 [cit. 2006-9-15]. Dostupné z: <http://www.microsoft.com>. Nápověda k programování v DirectX.

[8] MINAŘÍK, P. Mikrogen [online]. c2002, [cit. 2006-10-13]. <http://www.mikrogen.kobranet.cz/rs/index.php>.

[9] MIŽANIN, M. DirectX 8.1 [online]. [cit. 2007-3-14]. <http://www.mizanin.szm.sk/dx.html>.

[10] Pandasoft [online]. [cit. 2007-4-26]. <http://www.andytather.co.uk/Panda/directxmax_downloads.aspx>.

[11] PROJECT JEDI [online]. [cit. 2006-10-6]. <http://delphi-jedi.org>.

[12] ROOT.CZ – OpenGL a Direct3D [online]. 4.8.2004, [cit. 2007-5-6]. <http://www.root.cz/clanky/opengl-a-direct3d>.

[13] STROHAL, M., FRENZEL, P., TRÄNKLE, J. Tutoriál Direkt3D [online]. c2000-2007, [cit. 2006-11-4].<http://www.dsdt.info/tutorials/direct3d>.

[14] SVOBODA, L. 1001 typů a triků pro Delphi: hotová řešení. 2. vydání, Praha: Computer Press, 2002. ISBN 80-7226-529-6.

[15] UNGER, E. Osobní stránky [online]. [cit. 2006-10-19]. <http://ungerik.net/en>.

[16] WALNUM, C. Programujeme grafiku v Microsoft Direct3D. 1. vydání, Brno: Computer Press, 2004. ISBN 80-251-0136-3.

[17] Wikipedie, otevřená encyklopedie [online], [cit. 2007-4-22]. <http://cs.wikipedia.org>.